NetworkPkg[ALL]: NetworkPkg Package Document

Table of Contents

NetworkPkg Package Document.....1
0.96.....1
Package Basic Information.....1
Known Issue.....1
INDEX.....1

PCD.....1
INDEX.....1

PcdsDynamic.....1
INDEX.....1

common.....1
INDEX.....1

gEfiNetworkPkgTokenSpaceGuid.PcdDhcp6UidType.....1
PCD Information.....1

PcdsDynamicEx.....1
INDEX.....1

common.....1
INDEX.....1

gEfiNetworkPkgTokenSpaceGuid.PcdDhcp6UidType.....1
PCD Information.....1

PcdsFeatureFlag.....1
INDEX.....1

common.....1
INDEX.....1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecCertificateEnabled.....1
PCD Information.....1

PcdsFixedAtBuild.....1
INDEX.....1

common.....1
INDEX.....1

gEfiNetworkPkgTokenSpaceGuid.PcdDhcp6UidType.....1
PCD Information.....1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCaFile.....1
PCD Information.....1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCaFileSize.....1
PCD Information.....1

Table of Contents

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificate....1
PCD Information....1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificateKey....1
PCD Information....1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificateKeySize....1
PCD Information....1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificateSize....1
PCD Information....1

PcdsPatchableInModule....1
INDEX....1

common....1
INDEX....1

gEfiNetworkPkgTokenSpaceGuid.PcdDhcp6UidType....1
PCD Information....1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCaFile....1
PCD Information....1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCaFileSize....1
PCD Information....1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificate....1
PCD Information....1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificateKey....1
PCD Information....1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificateKeySize....1
PCD Information....1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificateSize....1
PCD Information....1

GUID....1
INDEX....1

common....1
INDEX....1

gEfiNetworkPkgTokenSpaceGuid....1
GUID Information....1

gIp6ConfigNvDataGuid....1
Refer to Header File....1
GUID Information....1

Table of Contents

gIScsiConfigGuid.....1
GUID Information.....1
Refer to Header File.....1

Modules.....1
INDEX.....1

Dhcp6Dxe.....1
Basic Module Information.....1
Pcds.....1
Protocols.....1
Library Classes.....2
Source Files.....2

DnsDxe.....1
Basic Module Information.....1
Protocols.....1
Library Classes.....2
Source Files.....2

HttpBootDxe.....1
Basic Module Information.....1
Protocols.....1
Library Classes.....1
Source Files.....2

HttpDxe.....1
Basic Module Information.....1
Protocols.....1
Library Classes.....1
Source Files.....2

HttpUtilitiesDxe.....1
Basic Module Information.....1
Protocols.....1
Library Classes.....1
Source Files.....1
Module Dependencies.....2

IfConfig6.....1
Basic Module Information.....1
Protocols.....1
Library Classes.....1
Source Files.....2

Ip6Dxe.....1
Basic Module Information.....1
Protocols.....1
Guids.....2
Library Classes.....2
Source Files.....2

Table of Contents

IpSecConfig.....1
Basic Module Information.....1
Protocols.....1
Library Classes.....1
Source Files.....2

IpSecDxe.....1
Basic Module Information.....1
Pcds.....1
Protocols.....1
Library Classes.....2
Source Files.....2

IScsiDxe.....1
Basic Module Information.....1
Protocols.....1
Guids.....2
Library Classes.....2
Source Files.....2

Mtftp6Dxe.....1
Basic Module Information.....1
Protocols.....1
Library Classes.....1
Source Files.....2

Ping6.....1
Basic Module Information.....1
Protocols.....1
Library Classes.....1
Source Files.....2

TcpDxe.....1
Basic Module Information.....1
Protocols.....1
Library Classes.....2
Source Files.....2

Udp6Dxe.....1
Basic Module Information.....1
Protocols.....1
Library Classes.....1
Source Files.....2

UefiPxeBcDxe.....1
Basic Module Information.....1
Pcds.....1
Protocols.....1
Guids.....2
Library Classes.....2
Source Files.....3

Table of Contents

VConfig.....1
Basic Module Information.....1
Protocols.....1
Library Classes.....1
Source Files.....2

Data Structures.....1

_ARG_LIST Struct Reference.....1
Data Fields.....1
Field Documentation.....1

_AUTH_ALGORITHM Struct Reference.....1
Data Fields.....1
Field Documentation.....1

_DHCP6_IA_CB Struct Reference.....1
Data Fields.....1
Field Documentation.....1

_DHCP6_INF_CB Struct Reference.....1
Data Fields.....1
Field Documentation.....1

_DHCP6_INSTANCE Struct Reference.....1
Data Fields.....1
Field Documentation.....1

_DHCP6_SERVICE Struct Reference.....1
Data Fields.....1
Field Documentation.....1

_DHCP6_TX_CB Struct Reference.....1
Data Fields.....1
Field Documentation.....1

_DNS_DRIVER_DATA Struct Reference.....1
Data Fields.....1
Field Documentation.....1

_DNS_FLAGS Union Reference.....1
Data Fields.....1
Field Documentation.....1

_DNS_INSTANCE Struct Reference.....1
Data Fields.....1
Field Documentation.....1

_DNS_SERVICE Struct Reference.....1
Data Fields.....1
Field Documentation.....1

Table of Contents

_EFI_AH_HEADER Struct Reference 1
Data Fields 1
Field Documentation 1

_EFI_ESP_HEADER Struct Reference 1
Data Fields 1
Field Documentation 1

_EFI_ESP_TAIL Struct Reference 1
Data Fields 1
Field Documentation 1

_ENCRYPT_ALGORITHM Struct Reference 1
Data Fields 1
Field Documentation 1

_HASH_ALGORITHM Struct Reference 1
Data Fields 1
Field Documentation 1

_HTTP_BOOT_PRIVATE_DATA Struct Reference 1
Data Fields 1
Field Documentation 1

_HTTP_PROTOCOL Struct Reference 1
Data Fields 1
Field Documentation 1

_HTTP_SERVICE Struct Reference 1
Data Fields 1
Field Documentation 1

_ICMP6_ECHO_REQUEST_REPLY Struct Reference 1
Data Fields 1
Field Documentation 1

_IFCONFIG6_INTERFACE_CB Struct Reference 1
Data Fields 1
Field Documentation 1

_IFCONFIG6_PRIVATE_DATA Struct Reference 1
Data Fields 1
Field Documentation 1

_IKE_EXCHANGE_INTERFACE Struct Reference 1
Data Fields 1
Field Documentation 1

_IP6_ADDRESS_INFO Struct Reference 1
Data Fields 1
Field Documentation 1

Table of Contents

_IP6_CONFIG_INSTANCE Struct Reference 1
Data Fields 1
Field Documentation 1

_IP6_DAD_ENTRY Struct Reference 1
Data Fields 1
Field Documentation 1

_IP6_DEFAULT_ROUTER Struct Reference 1
Data Fields 1
Field Documentation 1

_IP6_DELAY_JOIN_LIST Struct Reference 1
Data Fields 1
Field Documentation 1

_IP6_ETHE_ADDR_OPTION Struct Reference 1
Data Fields 1
Field Documentation 1

_IP6_FORM_CALLBACK_INFO Struct Reference 1
Data Fields 1
Field Documentation 1

_IP6_FRAGMENT_HEADER Struct Reference 1
Data Fields 1
Field Documentation 1

_IP6_INTERFACE Struct Reference 1
Data Fields 1
Field Documentation 1

_IP6_MTU_OPTION Struct Reference 1
Data Fields 1
Field Documentation 1

_IP6_NEIGHBOR_ENTRY Struct Reference 1
Data Fields 1
Field Documentation 1

_IP6_PREFIX_INFO_OPTION Struct Reference 1
Data Fields 1
Field Documentation 1

_IP6_PREFIX_LIST_ENTRY Struct Reference 1
Data Fields 1
Field Documentation 1

_IP6_PROTOCOL Struct Reference 1
Data Fields 1
Field Documentation 1

Table of Contents

_IP6_ROUTE_TABLE Struct Reference....1
Data Fields....1
Field Documentation....1

_IP6_ROUTING_HEADER Struct Reference....1
Data Fields....1
Field Documentation....1

_IP6_SERVICE Struct Reference....1
Data Fields....1
Field Documentation....2

_IPSEC_PAD_ENTRY Struct Reference....1
Data Fields....1
Field Documentation....1

_IPSEC_PRIVATE_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

_IPSEC_RECYCLE_CONTEXT Struct Reference....1
Data Fields....1
Field Documentation....1

_IPSEC_SAD_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

_IPSEC_SAD_ENTRY Struct Reference....1
Data Fields....1
Field Documentation....1

_IPSEC_SPD_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

_IPSEC_SPD_ENTRY Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_ADDTIONAL_HEADER Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_ASYNC_MESSAGE Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_ATTEMPT_CONFIG_NVDATA Struct Reference....1
Data Fields....1
Field Documentation....1

Table of Contents

_ISCSI_BASIC_HEADER Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_ISCSI_BI_EXP_READ_DATA_LEN_AHS Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_CHAP_AUTH_CONFIG_NVDATA Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_CHAP_AUTH_DATA Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_ISCSI_CONFIG_IFR_NVDATA Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_CONNECTION Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_DRIVER_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_FORM_CALLBACK_INFO Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_IN_BUFFER_CONTEXT Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_KEY_VALUE_PAIR Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_LOGIN_REQUEST Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_ISCSI_LOGIN_RESPONSE Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

Table of Contents

_ISCSI_LOGOUT_REQUEST Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_ISCSI_LOGOUT_RESPONSE Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_ISCSI_NOP_IN Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_ISCSI_NOP_OUT Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_ISCSI_PRIVATE_PROTOCOL Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_READY_TO_TRANSFER Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_ISCSI_REJECT Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_ISCSI_ROOT_PATH_FIELD Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_SCSI_DATA_IN Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_ISCSI_SCSI_DATA_OUT Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_ISCSI_SENSE_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

Table of Contents

_ISCSI_SESSION Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_SESSION_CONFIG_NVDATA Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_SNACK_REQUEST Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_ISCSI_TCB Struct Reference....1
Data Fields....1
Field Documentation....1

_ISCSI_TMF_REQUEST Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_ISCSI_TMF_RESPONSE Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_ISCSI_XFER_CONTEXT Struct Reference....1
Data Fields....1
Field Documentation....1

_MTFTP6_GETINFO_CONTEXT Struct Reference....1
Data Fields....1
Field Documentation....1

_MTFTP6_INSTANCE Struct Reference....1
Data Fields....1
Field Documentation....1

_MTFTP6_SERVICE Struct Reference....1
Data Fields....1
Field Documentation....1

_NET_PROTOCOL Union Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_PING6_ICMP6_TX_INFO Struct Reference....1
Data Fields....1
Field Documentation....1

Table of Contents

_PING6_PRIVATE_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

_PXEBC_PRIVATE_DATA Struct Reference....1
Data Fields....1
Field Documentation....2

_PXEBC_PRIVATE_PROTOCOL Struct Reference....1
Data Fields....1
Field Documentation....1

_PXEBC_VIRTUAL_NIC Struct Reference....1
Data Fields....1
Field Documentation....1

_SCSI_COMMAND Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_SCSI_RESPONSE Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_SOCK_BUFFER Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_SOCK_COMPLETION_TOKEN Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_SOCK_INIT_DATA Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_SOCK_IO_TOKEN Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_SOCK_TOKEN Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

Table of Contents

_TCP4_MODE_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

_TCP4_ROUTE_INFO Struct Reference....1
Data Fields....1
Field Documentation....1

_TCP6_MODE_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

_TCP_CONTROL_BLOCK Struct Reference....1
Data Fields....1
Detailed Description....3
Field Documentation....3

_TCP_HEARTBEAT_TIMER Struct Reference....1
Data Fields....1
Field Documentation....1

_TCP_OPTION Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_TCP_PEER Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_TCP_PROTO_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

_TCP_RSV_DATA Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_TCP_SEG Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

_TCP_SERVICE_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

_TCP_SOCKET Struct Reference....1
Data Fields....1
Detailed Description....2

Table of Contents

_TCP_SOCKET Struct Reference
Field Documentation....2

_UDP6_INSTANCE_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

_UDP6_RXDATA_WRAP Struct Reference....1
Data Fields....1
Field Documentation....1

_UDP6_SERVICE_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

_VAR_CHECK_ITEM Struct Reference....1
Data Fields....1
Field Documentation....1

DELETE_POLICY_ENTRY_CONTEXT Struct Reference....1
Data Fields....1
Field Documentation....1

DNS4_CACHE Struct Reference....1
Data Fields....1
Field Documentation....1

DNS4_SERVER_INFOR Struct Reference....1
Data Fields....1
Field Documentation....1

DNS4_SERVER_IP Struct Reference....1
Data Fields....1
Field Documentation....1

DNS4_TOKEN_ENTRY Struct Reference....1
Data Fields....1
Field Documentation....1

DNS6_CACHE Struct Reference....1
Data Fields....1
Field Documentation....1

DNS6_SERVER_INFOR Struct Reference....1
Data Fields....1
Field Documentation....1

DNS6_SERVER_IP Struct Reference....1
Data Fields....1
Field Documentation....1

Table of Contents

DNS6_TOKEN_ENTRY Struct Reference....1
Data Fields....1
Field Documentation....1

DNS_ANSWER_SECTION Struct Reference....1
Data Fields....1
Field Documentation....1

DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT Struct Reference....1
Data Fields....1
Field Documentation....1

DNS_HEADER Struct Reference....1
Data Fields....1
Field Documentation....1

DNS_QUERY_SECTION Struct Reference....1
Data Fields....1
Field Documentation....1

EDIT_POLICY_ENTRY_CONTEXT Struct Reference....1
Data Fields....1
Field Documentation....1

HASH_DATA_FRAGMENT Struct Reference....1
Data Fields....1
Field Documentation....1

HII_VENDOR_DEVICE_PATH Struct Reference....1
Data Fields....1
Detailed Description....1
Field Documentation....1

HTTP4_IO_CONFIG_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_CACHE_CONTENT Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_CALLBACK_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_DHCP4_OPTION_ARCH Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_DHCP4_OPTION_CLID Struct Reference....1
Data Fields....1
Field Documentation....1

Table of Contents

HTTP_BOOT_DHCP4_OPTION_ENTRY Union Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_DHCP4_OPTION_MAX_MESG_SIZE Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_DHCP4_OPTION_MESG Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_DHCP4_OPTION_PARA Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_DHCP4_OPTION_UNDI Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_DHCP4_OPTION_UUID Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_DHCP4_PACKET Union Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_DHCP4_PACKET_CACHE Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_DHCP_PACKET_CACHE Union Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_ENTITY_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_OPTION_BOOT_ITEM Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_URI_CONTENT Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_BOOT_VENDOR_OPTION Struct Reference....1
Data Fields....1
Field Documentation....1

Table of Contents

HTTP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_IO Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_IO_CONFIG_DATA Union Reference....1
Data Fields....1
Field Documentation....1

HTTP_IO_HEADER Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_IO_RESOPNSE_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_TCP_TOKEN_WRAP Struct Reference....1
Data Fields....1
Field Documentation....1

HTTP_TOKEN_WRAP Struct Reference....1
Data Fields....1
Field Documentation....1

IKE_ALG_GUID_INFO Struct Reference....1
Data Fields....1
Field Documentation....1

IKE_HEADER Struct Reference....1
Data Fields....1
Field Documentation....1

IKE_PACKET Struct Reference....1
Data Fields....1
Field Documentation....1

IKE_PAYLOAD Struct Reference....1
Data Fields....1
Field Documentation....1

IKE_SA_ATTR_UNION Union Reference....1
Data Fields....1
Field Documentation....1

IKE_SA_ATTRIBUTE Struct Reference....1
Data Fields....1
Field Documentation....1

Table of Contents

IKE_UDP_SERVICE Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_AUTH Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_CERT Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_CERT_REQ Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_CFG Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_CFG_ATTRIBUTES Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_CHILD_SA_SESSION Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_COMMON_PAYLOAD_HEADER Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_DELETE Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_DH_BUFFER Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_ENCRYPTED Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_ID Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_INFO_EXCHANGE_CONTEXT Struct Reference.....1
Data Fields.....1
Field Documentation.....1

Table of Contents

IKEV2_KEY_EXCHANGE Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_NONCE Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_NOTIFY Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_PACKET_HANDLER Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_PAD_LEN Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_PROPOSAL Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_PROPOSAL_DATA Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_SA Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_SA_DATA Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_SA_PARAMS Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_SA_SESSION Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_SESSION_COMMON Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_SESSION_KEYS Struct Reference.....1
Data Fields.....1
Field Documentation.....1

Table of Contents

IKEV2_TRANSFORM Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_TRANSFORM_DATA Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_TS Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IKEV2_VENDOR Struct Reference.....1
Data Fields.....1
Field Documentation.....1

INSERT_POLICY_ENTRY_CONTEXT Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IP6_ADDRESS_INFO_ENTRY Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IP6_ASSEMBLE_ENTRY Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IP6_ASSEMBLE_TABLE Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IP6_CLIP_INFO Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IP6_CONFIG_DATA Union Reference.....1
Data Fields.....1
Field Documentation.....1

IP6_CONFIG_DATA_ITEM Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IP6_CONFIG_DATA_RECORD Struct Reference.....1
Data Fields.....1
Field Documentation.....1

IP6_CONFIG_IFR_NVDATA Struct Reference.....1
Data Fields.....1
Detailed Description.....1
Field Documentation.....1

Table of Contents

IP6_CONFIG_NVDATA Struct Reference....1
Data Fields....1
Field Documentation....1

IP6_CONFIG_VARIABLE Struct Reference....1
Data Fields....1
Field Documentation....1

IP6_DESTROY_CHILD_BY_ADDR_CALLBACK_CONTEXT Struct Reference....1
Data Fields....1
Field Documentation....1

IP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT Struct Reference....1
Data Fields....1
Field Documentation....1

IP6_IPSEC_WRAP Struct Reference....1
Data Fields....1
Field Documentation....1

IP6_LINK_RX_TOKEN Struct Reference....1
Data Fields....1
Field Documentation....1

IP6_LINK_TX_TOKEN Struct Reference....1
Data Fields....1
Field Documentation....1

IP6_MLD_GROUP Struct Reference....1
Data Fields....1
Field Documentation....1

IP6_MLD_HEAD Struct Reference....1
Data Fields....1
Field Documentation....1

IP6_MLD_SERVICE_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

IP6_ROUTE_CACHE Struct Reference....1
Data Fields....1
Field Documentation....1

IP6_ROUTE_CACHE_ENTRY Struct Reference....1
Data Fields....1
Field Documentation....1

IP6_ROUTE_ENTRY Struct Reference....1
Data Fields....1
Field Documentation....1

Table of Contents

IP6_RXDATA_WRAP Struct Reference....1
Data Fields....1
Field Documentation....1

IP6_TXTOKEN_WRAP Struct Reference....1
Data Fields....1
Field Documentation....1

IP_SEC_VARIABLE_INFO Struct Reference....1
Data Fields....1
Field Documentation....1

IPSEC_COMMON_POLICY_ENTRY Struct Reference....1
Data Fields....1
Field Documentation....1

IPSEC_VAR_ITEM_HEADER Struct Reference....1
Data Fields....1
Field Documentation....1

IPSEC_VARIABLE_BUFFER Struct Reference....1
Data Fields....1
Field Documentation....1

ISCSI_NIC_INFO Struct Reference....1
Data Fields....1
Field Documentation....1

ISCSI_PRIVATE_DATA Struct Reference....1
Data Fields....1
Field Documentation....1

MODP_GROUP Struct Reference....1
Data Fields....1
Field Documentation....1

MTFTP6_BLOCK_RANGE Struct Reference....1
Data Fields....1
Field Documentation....1

MTFTP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT Struct Reference....1
Data Fields....1
Field Documentation....1

MTFTP6_EXT_OPTION_INFO Struct Reference....1
Data Fields....1
Field Documentation....1

PAD_ENTRY_INDEXER Struct Reference....1
Data Fields....1
Field Documentation....1

Table of Contents

POLICY_ENTRY_INDEXER Union Reference 1
Data Fields 1
Field Documentation 1

PRF_DATA_FRAGMENT Struct Reference 1
Data Fields 1
Field Documentation 1

PXEBC_BOOT_MENU_ENTRY Struct Reference 1
Data Fields 1
Field Documentation 1

PXEBC_BOOT_SVR_ENTRY Struct Reference 1
Data Fields 1
Field Documentation 1

PXEBC_CLASS_ID Struct Reference 1
Data Fields 1
Field Documentation 1

PXEBC_DHCP4_OPTION_ARCH Struct Reference 1
Data Fields 1
Field Documentation 1

PXEBC_DHCP4_OPTION_CLID Struct Reference 1
Data Fields 1
Field Documentation 1

PXEBC_DHCP4_OPTION_ENTRY Union Reference 1
Data Fields 1
Field Documentation 1

PXEBC_DHCP4_OPTION_MAX_MESG_SIZE Struct Reference 1
Data Fields 1
Field Documentation 1

PXEBC_DHCP4_OPTION_MESG Struct Reference 1
Data Fields 1
Field Documentation 1

PXEBC_DHCP4_OPTION_PARA Struct Reference 1
Data Fields 1
Field Documentation 1

PXEBC_DHCP4_OPTION_UNDI Struct Reference 1
Data Fields 1
Field Documentation 1

PXEBC_DHCP4_OPTION_UUID Struct Reference 1
Data Fields 1
Field Documentation 1

Table of Contents

PXEBC_DHCP4_PACKET Union Reference....1
Data Fields....1
Field Documentation....1

PXEBC_DHCP4_PACKET_CACHE Struct Reference....1
Data Fields....1
Field Documentation....1

PXEBC_DHCP6_OPTION_ARCH Struct Reference....1
Data Fields....1
Field Documentation....1

PXEBC_DHCP6_OPTION_ENTRY Union Reference....1
Data Fields....1
Field Documentation....1

PXEBC_DHCP6_OPTION_NODE Struct Reference....1
Data Fields....1
Field Documentation....1

PXEBC_DHCP6_OPTION_ORO Struct Reference....1
Data Fields....1
Field Documentation....1

PXEBC_DHCP6_OPTION_UNDI Struct Reference....1
Data Fields....1
Field Documentation....1

PXEBC_DHCP6_OPTION_VENDOR_CLASS Struct Reference....1
Data Fields....1
Field Documentation....1

PXEBC_DHCP6_PACKET Union Reference....1
Data Fields....1
Field Documentation....1

PXEBC_DHCP6_PACKET_CACHE Struct Reference....1
Data Fields....1
Field Documentation....1

PXEBC_DHCP_PACKET_CACHE Union Reference....1
Data Fields....1
Field Documentation....1

PXEBC_MENU_PROMPT Struct Reference....1
Data Fields....1
Field Documentation....1

PXEBC_OPTION_BOOT_ITEM Struct Reference....1
Data Fields....1
Field Documentation....1

Table of Contents

PXEBC_VENDOR_OPTION Struct Reference.....1
Data Fields.....1
Field Documentation.....1

SA_KEYMATS Struct Reference.....1
Data Fields.....1
Field Documentation.....1

SAD_ENTRY_INDEXER Struct Reference.....1
Data Fields.....1
Field Documentation.....1

SHELL_PARAM_PACKAGE Struct Reference.....1
Data Fields.....1
Field Documentation.....1

SOCK_IO_DATA Union Reference.....1
Data Fields.....1
Field Documentation.....1

SPD_ENTRY_INDEXER Struct Reference.....1
Data Fields.....1
Field Documentation.....1

STR2INT Struct Reference.....1
Data Fields.....1
Field Documentation.....1

TCP_ACCESS_POINT Union Reference.....1
Data Fields.....1
Field Documentation.....1

TCP_CONFIG_DATA Union Reference.....1
Data Fields.....1
Field Documentation.....1

TCP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT Struct Reference.....1
Data Fields.....1
Field Documentation.....1

TRAFFIC_SELECTOR Struct Reference.....1
Data Fields.....1
Field Documentation.....1

UDP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT Struct Reference.....1
Data Fields.....1
Field Documentation.....1

VAR_CHECK_ITEM Struct Reference.....1
Data Fields.....1
Field Documentation.....1
- a -.....2

Table of Contents

File List 1

NetworkPkg/Application/IfConfig6/IfConfig6.c File Reference 1
- Functions 1
- Variables 1
- Detailed Description 1
- Function Documentation 2
- Variable Documentation 7

NetworkPkg/Application/IfConfig6/IfConfig6.h File Reference 1
- Data Structures 1
- Defines 1
- Typedefs 1
- Enumerations 1
- Detailed Description 1
- Define Documentation 2
- Typedef Documentation 2
- Enumeration Type Documentation 2

NetworkPkg/Application/IpsecConfig/Delete.c File Reference 1
- Functions 1
- Detailed Description 1
- Function Documentation 1

NetworkPkg/Application/IpsecConfig/Delete.h File Reference 1
- Data Structures 1
- Functions 1
- Detailed Description 1
- Function Documentation 1

NetworkPkg/Application/IpsecConfig/Dump.c File Reference 1
- Functions 1
- Variables 1
- Detailed Description 1
- Function Documentation 1
- Variable Documentation 4

NetworkPkg/Application/IpsecConfig/Dump.h File Reference 1
- Functions 1
- Detailed Description 1
- Function Documentation 1

NetworkPkg/Application/IpsecConfig/ForEach.c File Reference 1
- Functions 1
- Detailed Description 1
- Function Documentation 1

NetworkPkg/Application/IpsecConfig/ForEach.h File Reference 1
- Typedefs 1
- Functions 1
- Detailed Description 1
- Typedef Documentation 1

Table of Contents

NetworkPkg/Application/IpsecConfig/ForEach.h File Reference
Function Documentation....1

NetworkPkg/Application/IpsecConfig/Helper.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/Application/IpsecConfig/Helper.h File Reference....1
Defines....1
Functions....1
Detailed Description....1
Define Documentation....1
Function Documentation....2

NetworkPkg/Application/IpsecConfig/Indexer.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....2

NetworkPkg/Application/IpsecConfig/Indexer.h File Reference....1
Data Structures....1
Typedefs....1
Variables....1
Detailed Description....1
Typedef Documentation....1
Variable Documentation....2

NetworkPkg/Application/IpsecConfig/IpSecConfig.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....4

NetworkPkg/Application/IpsecConfig/IpSecConfig.h File Reference....1
Data Structures....1
Defines....1
Variables....1
Detailed Description....2
Define Documentation....2
Variable Documentation....4

NetworkPkg/Application/IpsecConfig/Match.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....3

Table of Contents

NetworkPkg/Application/IpsecConfig/Match.h File Reference....1
Typedefs....1
Variables....1
Detailed Description....1
Typedef Documentation....1
Variable Documentation....1

NetworkPkg/Application/IpsecConfig/PolicyEntryOperation.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....7

NetworkPkg/Application/IpsecConfig/PolicyEntryOperation.h File Reference....1
Data Structures....1
Defines....1
Typedefs....2
Functions....2
Detailed Description....2
Define Documentation....2
Typedef Documentation....6
Function Documentation....7

NetworkPkg/Application/Ping6/Ping6.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....6

NetworkPkg/Application/Ping6/Ping6.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Detailed Description....1
Define Documentation....1
Typedef Documentation....2
Function Documentation....2

NetworkPkg/Application/Ping6/Ia32/Tsc.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/Application/Ping6/Ipf/Itc.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

Table of Contents

NetworkPkg/Application/Ping6/X64/Tsc.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/Application/VConfig/VConfig.c File Reference....1
Defines....1
Functions....1
Variables....1
Detailed Description....1
Define Documentation....2
Function Documentation....2
Variable Documentation....5

NetworkPkg/Dhcp6Dxe/ComponentName.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....4

NetworkPkg/Dhcp6Dxe/Dhcp6Driver.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....6

NetworkPkg/Dhcp6Dxe/Dhcp6Driver.h File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....5

NetworkPkg/Dhcp6Dxe/Dhcp6Impl.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....8

NetworkPkg/Dhcp6Dxe/Dhcp6Impl.h File Reference....1
Data Structures....1
Defines....1
Typedefs....2
Enumerations....2
Functions....3
Variables....4
Detailed Description....4
Define Documentation....4
Typedef Documentation....8

Table of Contents

NetworkPkg/Dhcp6Dxe/Dhcp6Impl.h File Reference
Enumeration Type Documentation....9
Function Documentation....10
Variable Documentation....20

NetworkPkg/Dhcp6Dxe/Dhcp6Io.c File Reference....1
Functions....1
Detailed Description....2
Function Documentation....2

NetworkPkg/Dhcp6Dxe/Dhcp6Io.h File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/Dhcp6Dxe/Dhcp6Utility.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....2

NetworkPkg/Dhcp6Dxe/Dhcp6Utility.h File Reference....1
Defines....1
Functions....1
Detailed Description....2
Define Documentation....2
Function Documentation....2

NetworkPkg/DnsDxe/ComponentName.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....3

NetworkPkg/DnsDxe/DnsDhcp.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/DnsDxe/DnsDhcp.h File Reference....1
Data Structures....1
Defines....1
Functions....1
Detailed Description....1
Define Documentation....2
Function Documentation....3

NetworkPkg/DnsDxe/DnsDriver.c File Reference....1
Functions....1
Variables....2
Detailed Description....2
Function Documentation....2

Table of Contents

NetworkPkg/DnsDxe/DnsDriver.c File Reference
Variable Documentation....12

NetworkPkg/DnsDxe/DnsDriver.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Variables....2
Detailed Description....2
Define Documentation....3
Typedef Documentation....3
Function Documentation....4
Variable Documentation....13

NetworkPkg/DnsDxe/DnsImpl.c File Reference....1
Functions....1
Detailed Description....2
Function Documentation....2

NetworkPkg/DnsDxe/DnsImpl.h File Reference....1
Data Structures....1
Defines....1
Typedefs....2
Functions....2
Variables....4
Detailed Description....4
Define Documentation....5
Typedef Documentation....7
Function Documentation....7
Variable Documentation....26

NetworkPkg/DnsDxe/DnsProtocol.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....10

NetworkPkg/HttpBootDxe/HttpBootClient.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/HttpBootDxe/HttpBootClient.h File Reference....1
Data Structures....1
Defines....1
Functions....1
Detailed Description....1
Define Documentation....1
Function Documentation....2

Table of Contents

NetworkPkg/HttpBootDxe/HttpBootComponentName.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....3

NetworkPkg/HttpBootDxe/HttpBootComponentName.h File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/HttpBootDxe/HttpBootDhcp4.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....6

NetworkPkg/HttpBootDxe/HttpBootDhcp4.h File Reference....1
Data Structures....1
Defines....1
Enumerations....2
Functions....2
Detailed Description....3
Define Documentation....3
Enumeration Type Documentation....9
Function Documentation....9

NetworkPkg/HttpBootDxe/HttpBootDxe.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....5

NetworkPkg/HttpBootDxe/HttpBootDxe.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Variables....1
Detailed Description....2
Define Documentation....2
Typedef Documentation....2
Function Documentation....2
Variable Documentation....5

NetworkPkg/HttpBootDxe/HttpBootImpl.c File Reference....1
Functions....1
Variables....1
Detailed Description....1

Table of Contents

NetworkPkg/HttpBootDxe/HttpBootImpl.c File Reference
Function Documentation....1
Variable Documentation....4

NetworkPkg/HttpBootDxe/HttpBootImpl.h File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/HttpBootDxe/HttpBootSupport.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/HttpBootDxe/HttpBootSupport.h File Reference....1
Data Structures....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/HttpDxe/ComponentName.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....3

NetworkPkg/HttpDxe/ComponentName.h File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/HttpDxe/HttpDns.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/HttpDxe/HttpDns.h File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/HttpDxe/HttpDriver.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....7

NetworkPkg/HttpDxe/HttpDriver.h File Reference....1
Data Structures....1
Defines....1

Table of Contents

NetworkPkg/HttpDxe/HttpDriver.h File Reference
Functions....1
Variables....1
Detailed Description....1
Define Documentation....2
Function Documentation....2
Variable Documentation....6

NetworkPkg/HttpDxe/HttpImpl.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....7

NetworkPkg/HttpDxe/HttpImpl.h File Reference....1
Defines....1
Functions....1
Variables....1
Detailed Description....1
Define Documentation....2
Function Documentation....2
Variable Documentation....7

NetworkPkg/HttpDxe/HttpProto.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....2

NetworkPkg/HttpDxe/HttpProto.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Detailed Description....2
Define Documentation....3
Typedef Documentation....5
Function Documentation....5

NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesDxe.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesDxe.h File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....5

Table of Contents

NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesImpl.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesProtocol.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....3

NetworkPkg/Include/Guid/Ip6ConfigHii.h File Reference....1
Defines....1
Variables....1
Detailed Description....1
Define Documentation....1
Variable Documentation....1

NetworkPkg/Include/Guid/IScsiConfigHii.h File Reference....1
Defines....1
Variables....1
Detailed Description....1
Define Documentation....1
Variable Documentation....1

NetworkPkg/Ip6Dxe/ComponentName.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....4

NetworkPkg/Ip6Dxe/Ip6Common.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/Ip6Dxe/Ip6Common.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Enumerations....1
Functions....1
Detailed Description....2
Define Documentation....2
Typedef Documentation....4
Enumeration Type Documentation....4
Function Documentation....4

Table of Contents

NetworkPkg/Ip6Dxe/Ip6Config.vfr File Reference....1
Defines....1
Variables....1
Detailed Description....1
Define Documentation....2
Variable Documentation....2

NetworkPkg/Ip6Dxe/Ip6ConfigImpl.c File Reference....1
Functions....1
Variables....2
Detailed Description....2
Function Documentation....2
Variable Documentation....15

NetworkPkg/Ip6Dxe/Ip6ConfigImpl.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Detailed Description....2
Define Documentation....2
Typedef Documentation....4
Function Documentation....5

NetworkPkg/Ip6Dxe/Ip6ConfigNv.c File Reference....1
Functions....1
Variables....1
Detailed Description....2
Function Documentation....2
Variable Documentation....11

NetworkPkg/Ip6Dxe/Ip6ConfigNv.h File Reference....1
Defines....1
Enumerations....1
Functions....1
Variables....1
Detailed Description....1
Define Documentation....2
Enumeration Type Documentation....2
Function Documentation....2
Variable Documentation....3

NetworkPkg/Ip6Dxe/Ip6Driver.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....6

NetworkPkg/Ip6Dxe/Ip6Driver.h File Reference....1
Data Structures....1
Functions....1

Table of Contents

NetworkPkg/Ip6Dxe/Ip6Driver.h File Reference
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....5

NetworkPkg/Ip6Dxe/Ip6Icmp.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....5

NetworkPkg/Ip6Dxe/Ip6Icmp.h File Reference....1
Defines....1
Functions....1
Variables....1
Detailed Description....1
Define Documentation....1
Function Documentation....2
Variable Documentation....3

NetworkPkg/Ip6Dxe/Ip6If.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/Ip6Dxe/Ip6If.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Detailed Description....2
Define Documentation....2
Typedef Documentation....2
Function Documentation....3

NetworkPkg/Ip6Dxe/Ip6Impl.c File Reference....1
Functions....1
Variables....1
Detailed Description....2
Function Documentation....2
Variable Documentation....14

NetworkPkg/Ip6Dxe/Ip6Impl.h File Reference....1
Data Structures....1
Defines....1
Functions....1
Variables....2
Detailed Description....2
Define Documentation....2
Function Documentation....3

Table of Contents

NetworkPkg/Ip6Dxe/Ip6Impl.h File Reference
Variable Documentation....20

NetworkPkg/Ip6Dxe/Ip6Input.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....2

NetworkPkg/Ip6Dxe/Ip6Input.h File Reference....1
Data Structures....1
Defines....1
Functions....1
Detailed Description....2
Define Documentation....2
Function Documentation....3

NetworkPkg/Ip6Dxe/Ip6Mld.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/Ip6Dxe/Ip6Mld.h File Reference....1
Data Structures....1
Defines....1
Functions....1
Detailed Description....1
Define Documentation....2
Function Documentation....2

NetworkPkg/Ip6Dxe/Ip6Nd.c File Reference....1
Functions....1
Variables....2
Detailed Description....2
Function Documentation....2
Variable Documentation....16

NetworkPkg/Ip6Dxe/Ip6Nd.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Enumerations....1
Functions....2
Detailed Description....4
Define Documentation....4
Typedef Documentation....5
Enumeration Type Documentation....5
Function Documentation....6

NetworkPkg/Ip6Dxe/Ip6NvData.h File Reference....1
Data Structures....1
Defines....1
Detailed Description....1

Table of Contents

NetworkPkg/Ip6Dxe/Ip6NvData.h File Reference
Define Documentation....2

NetworkPkg/Ip6Dxe/Ip6Option.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/Ip6Dxe/Ip6Option.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Enumerations....1
Functions....1
Detailed Description....2
Define Documentation....2
Typedef Documentation....2
Enumeration Type Documentation....3
Function Documentation....3

NetworkPkg/Ip6Dxe/Ip6Output.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....6

NetworkPkg/Ip6Dxe/Ip6Output.h File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....4

NetworkPkg/Ip6Dxe/Ip6Route.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....2

NetworkPkg/Ip6Dxe/Ip6Route.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Detailed Description....2
Define Documentation....2
Typedef Documentation....3
Function Documentation....3

NetworkPkg/IpSecDxe/ComponentName.c File Reference....1
Functions....1
Variables....1

Table of Contents

NetworkPkg/IpSecDxe/ComponentName.c File Reference
Detailed Description....1
Function Documentation....1
Variable Documentation....4

NetworkPkg/IpSecDxe/IetfConstants.c File Reference....1
Defines....1
Variables....1
Detailed Description....1
Define Documentation....1
Variable Documentation....1

NetworkPkg/IpSecDxe/Ike.h File Reference....1
Data Structures....1
Defines....1
Typedefs....2
Enumerations....2
Detailed Description....2
Define Documentation....3
Typedef Documentation....6
Enumeration Type Documentation....6

NetworkPkg/IpSecDxe/IkeCommon.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....4

NetworkPkg/IpSecDxe/IkeCommon.h File Reference....1
Defines....1
Functions....1
Variables....1
Detailed Description....1
Define Documentation....2
Function Documentation....3
Variable Documentation....6

NetworkPkg/IpSecDxe/IkePacket.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/IpSecDxe/IkePacket.h File Reference....1
Defines....1
Functions....1
Detailed Description....1
Define Documentation....1
Function Documentation....1

Table of Contents

NetworkPkg/IpSecDxe/IkeService.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....6

NetworkPkg/IpSecDxe/IkeService.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Variables....2
Detailed Description....2
Define Documentation....2
Typedef Documentation....2
Function Documentation....4
Variable Documentation....8

NetworkPkg/IpSecDxe/IpSecConfigImpl.c File Reference....1
Functions....1
Variables....2
Detailed Description....3
Function Documentation....3
Variable Documentation....19

NetworkPkg/IpSecDxe/IpSecConfigImpl.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....2
Variables....3
Detailed Description....3
Define Documentation....4
Typedef Documentation....5
Function Documentation....7
Variable Documentation....28

NetworkPkg/IpSecDxe/IpSecCryptIo.c File Reference....1
Functions....1
Variables....2
Detailed Description....2
Function Documentation....2
Variable Documentation....11

NetworkPkg/IpSecDxe/IpSecCryptIo.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....2
Detailed Description....3
Define Documentation....3

Table of Contents

NetworkPkg/IpSecDxe/IpSecCryptIo.h File Reference
Typedef Documentation....4
Function Documentation....8

NetworkPkg/IpSecDxe/IpSecDebug.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....3

NetworkPkg/IpSecDxe/IpSecDebug.h File Reference....1
Defines....1
Functions....1
Detailed Description....1
Define Documentation....2
Function Documentation....3

NetworkPkg/IpSecDxe/IpSecDriver.c File Reference....1
Functions....1
Variables....1
Detailed Description....2
Function Documentation....2
Variable Documentation....7

NetworkPkg/IpSecDxe/IpSecImpl.c File Reference....1
Functions....1
Detailed Description....2
Function Documentation....2

NetworkPkg/IpSecDxe/IpSecImpl.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....2
Variables....3
Detailed Description....3
Define Documentation....3
Typedef Documentation....4
Function Documentation....5
Variable Documentation....11

NetworkPkg/IpSecDxe/IpSecMain.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....2

NetworkPkg/IpSecDxe/Ikev2/ChildSa.c File Reference....1
Functions....1
Variables....1

Table of Contents

NetworkPkg/IpSecDxe/Ikev2/ChildSa.c File Reference
Detailed Description....1
Function Documentation....1
Variable Documentation....3

NetworkPkg/IpSecDxe/Ikev2/Exchange.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....5

NetworkPkg/IpSecDxe/Ikev2/Ikev2.h File Reference....1
Data Structures....1
Defines....1
Typedefs....2
Enumerations....2
Variables....2
Detailed Description....2
Define Documentation....3
Typedef Documentation....5
Enumeration Type Documentation....5
Variable Documentation....6

NetworkPkg/IpSecDxe/Ikev2/Info.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....2

NetworkPkg/IpSecDxe/Ikev2/Payload.c File Reference....1
Defines....1
Functions....1
Variables....3
Detailed Description....3
Define Documentation....3
Function Documentation....3
Variable Documentation....17

NetworkPkg/IpSecDxe/Ikev2/Payload.h File Reference....1
Data Structures....1
Defines....1
Detailed Description....3
Define Documentation....4

NetworkPkg/IpSecDxe/Ikev2/Sa.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....8

Table of Contents

NetworkPkg/IpSecDxe/Ikev2/Utility.c File Reference....1
Functions....1
Variables....2
Detailed Description....3
Function Documentation....3
Variable Documentation....16

NetworkPkg/IpSecDxe/Ikev2/Utility.h File Reference....1
Defines....1
Functions....1
Variables....4
Detailed Description....4
Define Documentation....4
Function Documentation....5
Variable Documentation....38

NetworkPkg/IScsiDxe/ComponentName.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....4

NetworkPkg/IScsiDxe/IScsiAuthenticationInfo.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....2

NetworkPkg/IScsiDxe/IScsiCHAP.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/IScsiDxe/IScsiCHAP.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Detailed Description....1
Define Documentation....2
Typedef Documentation....3
Function Documentation....3

NetworkPkg/IScsiDxe/IScsiConfig.c File Reference....1
Functions....1
Variables....2
Detailed Description....2
Function Documentation....2
Variable Documentation....12

Table of Contents

NetworkPkg/IScsiDxe/IScsiConfig.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Variables....1
Detailed Description....2
Define Documentation....2
Typedef Documentation....3
Function Documentation....3
Variable Documentation....5

NetworkPkg/IScsiDxe/IScsiConfigNVDataStruc.h File Reference....1
Data Structures....1
Defines....1
Typedefs....2
Detailed Description....3
Define Documentation....3
Typedef Documentation....11

NetworkPkg/IScsiDxe/IScsiConfigVfr.vfr File Reference....1
Defines....1
Variables....1
Detailed Description....2
Define Documentation....2
Variable Documentation....2

NetworkPkg/IScsiDxe/IScsiDhcp.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/IScsiDxe/IScsiDhcp.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Detailed Description....1
Define Documentation....2
Typedef Documentation....3
Function Documentation....3

NetworkPkg/IScsiDxe/IScsiDhcp6.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/IScsiDxe/IScsiDhcp6.h File Reference....1
Defines....1
Functions....1
Detailed Description....1
Define Documentation....1

Table of Contents

NetworkPkg/IScsiDxe/IScsiDhcp6.h File Reference
Function Documentation....2

NetworkPkg/IScsiDxe/IScsiDriver.c File Reference....1
Functions....1
Variables....1
Detailed Description....2
Function Documentation....2
Variable Documentation....10

NetworkPkg/IScsiDxe/IScsiDriver.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Variables....2
Detailed Description....3
Define Documentation....3
Typedef Documentation....3
Function Documentation....3
Variable Documentation....19

NetworkPkg/IScsiDxe/IScsiExtScsiPassThru.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....6

NetworkPkg/IScsiDxe/IScsiIbft.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....4

NetworkPkg/IScsiDxe/IScsiIbft.h File Reference....1
Defines....1
Functions....1
Detailed Description....1
Define Documentation....1
Function Documentation....2

NetworkPkg/IScsiDxe/IScsiImpl.h File Reference....1
Data Structures....1
Defines....1
Detailed Description....1
Define Documentation....1

NetworkPkg/IScsiDxe/IScsiInitiatorName.c File Reference....1
Functions....1
Variables....1

Table of Contents

NetworkPkg/IScsiDxe/IScsiInitiatorName.c File Reference
Detailed Description....1
Function Documentation....1
Variable Documentation....2

NetworkPkg/IScsiDxe/IScsiMisc.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....10

NetworkPkg/IScsiDxe/IScsiMisc.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Detailed Description....2
Define Documentation....2
Typedef Documentation....3
Function Documentation....3

NetworkPkg/IScsiDxe/IScsiProto.c File Reference....1
Functions....1
Variables....2
Detailed Description....2
Function Documentation....2
Variable Documentation....19

NetworkPkg/IScsiDxe/IScsiProto.h File Reference....1
Data Structures....1
Defines....1
Typedefs....4
Enumerations....5
Functions....5
Detailed Description....5
Define Documentation....6
Typedef Documentation....18
Enumeration Type Documentation....20
Function Documentation....20

NetworkPkg/Mtftp6Dxe/ComponentName.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....4

NetworkPkg/Mtftp6Dxe/Mtftp6Driver.c File Reference....1
Functions....1
Variables....1
Detailed Description....1

Table of Contents

NetworkPkg/Mtftp6Dxe/Mtftp6Driver.c File Reference
Function Documentation....2
Variable Documentation....6

NetworkPkg/Mtftp6Dxe/Mtftp6Driver.h File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....4

NetworkPkg/Mtftp6Dxe/Mtftp6Impl.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....8

NetworkPkg/Mtftp6Dxe/Mtftp6Impl.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Variables....2
Detailed Description....2
Define Documentation....2
Typedef Documentation....3
Function Documentation....3
Variable Documentation....14

NetworkPkg/Mtftp6Dxe/Mtftp6Option.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....3

NetworkPkg/Mtftp6Dxe/Mtftp6Option.h File Reference....1
Data Structures....1
Defines....1
Functions....1
Variables....1
Detailed Description....1
Define Documentation....2
Function Documentation....3
Variable Documentation....5

NetworkPkg/Mtftp6Dxe/Mtftp6Rrq.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

Table of Contents

NetworkPkg/Mtftp6Dxe/Mtftp6Support.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....2

NetworkPkg/Mtftp6Dxe/Mtftp6Support.h File Reference....1
Data Structures....1
Functions....1
Detailed Description....1
Function Documentation....2

NetworkPkg/Mtftp6Dxe/Mtftp6Wrq.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/TcpDxe/ComponentName.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....4

NetworkPkg/TcpDxe/Socket.h File Reference....1
Data Structures....1
Defines....1
Typedefs....2
Enumerations....3
Functions....3
Detailed Description....3
Define Documentation....4
Typedef Documentation....12
Enumeration Type Documentation....14
Function Documentation....14

NetworkPkg/TcpDxe/SockImpl.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....2

NetworkPkg/TcpDxe/SockImpl.h File Reference....1
Defines....1
Functions....1
Detailed Description....1
Define Documentation....1
Function Documentation....2

NetworkPkg/TcpDxe/SockInterface.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

Table of Contents

NetworkPkg/TcpDxe/TcpDispatcher.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/TcpDxe/TcpDriver.c File Reference....1
Functions....1
Variables....2
Detailed Description....2
Function Documentation....2
Variable Documentation....8

NetworkPkg/TcpDxe/TcpDriver.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Detailed Description....2
Define Documentation....2
Typedef Documentation....3
Function Documentation....3

NetworkPkg/TcpDxe/TcpFunc.h File Reference....1
Defines....1
Typedefs....1
Functions....1
Detailed Description....3
Define Documentation....3
Typedef Documentation....4
Function Documentation....4

NetworkPkg/TcpDxe/TcpInput.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/TcpDxe/TcpIo.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/TcpDxe/TcpMain.c File Reference....1
Functions....1
Detailed Description....2
Function Documentation....2

NetworkPkg/TcpDxe/TcpMain.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Variables....2

Table of Contents

NetworkPkg/TcpDxe/TcpMain.h File Reference
Detailed Description....2
Define Documentation....3
Typedef Documentation....3
Function Documentation....3
Variable Documentation....23

NetworkPkg/TcpDxe/TcpMisc.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....8

NetworkPkg/TcpDxe/TcpOption.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/TcpDxe/TcpOption.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Detailed Description....2
Define Documentation....2
Typedef Documentation....4
Function Documentation....4

NetworkPkg/TcpDxe/TcpOutput.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....7

NetworkPkg/TcpDxe/TcpProto.h File Reference....1
Data Structures....1
Defines....1
Typedefs....4
Detailed Description....4
Define Documentation....4
Typedef Documentation....15

NetworkPkg/TcpDxe/TcpTimer.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....5

Table of Contents

NetworkPkg/Udp6Dxe/ComponentName.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....4

NetworkPkg/Udp6Dxe/Udp6Driver.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....5

NetworkPkg/Udp6Dxe/Udp6Driver.h File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/Udp6Dxe/Udp6Impl.c File Reference....1
Functions....1
Variables....2
Detailed Description....2
Function Documentation....2
Variable Documentation....13

NetworkPkg/Udp6Dxe/Udp6Impl.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Functions....1
Variables....2
Detailed Description....2
Define Documentation....3
Typedef Documentation....4
Function Documentation....4
Variable Documentation....18

NetworkPkg/Udp6Dxe/Udp6Main.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....6

NetworkPkg/UefiPxeBcDxe/ComponentName.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....1
Variable Documentation....4

Table of Contents

NetworkPkg/UefiPxeBcDxe/PxeBcBoot.c File Reference....1
Functions....1
Detailed Description....1
Function Documentation....1

NetworkPkg/UefiPxeBcDxe/PxeBcBoot.h File Reference....1
Defines....1
Functions....1
Detailed Description....1
Define Documentation....1
Function Documentation....2

NetworkPkg/UefiPxeBcDxe/PxeBcDhcp4.c File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....9

NetworkPkg/UefiPxeBcDxe/PxeBcDhcp4.h File Reference....1
Data Structures....1
Defines....1
Enumerations....3
Functions....3
Detailed Description....3
Define Documentation....4
Enumeration Type Documentation....13
Function Documentation....13

NetworkPkg/UefiPxeBcDxe/PxeBcDhcp6.c File Reference....1
Functions....1
Variables....2
Detailed Description....2
Function Documentation....2
Variable Documentation....11

NetworkPkg/UefiPxeBcDxe/PxeBcDhcp6.h File Reference....1
Data Structures....1
Defines....1
Functions....2
Detailed Description....2
Define Documentation....3
Function Documentation....6

NetworkPkg/UefiPxeBcDxe/PxeBcDriver.c File Reference....1
Functions....1
Variables....2
Detailed Description....2
Function Documentation....2
Variable Documentation....9

Table of Contents

NetworkPkg/UefiPxeBcDxe/PxeBcDriver.h File Reference....1
Functions....1
Variables....1
Detailed Description....1
Function Documentation....2
Variable Documentation....5

NetworkPkg/UefiPxeBcDxe/PxeBcImpl.c File Reference....1
Functions....1
Variables....2
Detailed Description....2
Function Documentation....3
Variable Documentation....15

NetworkPkg/UefiPxeBcDxe/PxeBcImpl.h File Reference....1
Data Structures....1
Defines....1
Typedefs....1
Variables....1
Detailed Description....2
Define Documentation....2
Typedef Documentation....3
Variable Documentation....4

NetworkPkg/UefiPxeBcDxe/PxeBcMtftp.c File Reference....1
Functions....1
Variables....2
Detailed Description....2
Function Documentation....2
Variable Documentation....10

NetworkPkg/UefiPxeBcDxe/PxeBcMtftp.h File Reference....1
Defines....1
Functions....1
Detailed Description....1
Define Documentation....2
Function Documentation....2

NetworkPkg/UefiPxeBcDxe/PxeBcSupport.c File Reference....1
Functions....1
Detailed Description....2
Function Documentation....2

NetworkPkg/UefiPxeBcDxe/PxeBcSupport.h File Reference....1
Defines....1
Functions....1
Detailed Description....3
Define Documentation....3
Function Documentation....4
- a -....18

NetworkPkg Package Document

0.96

Network Package.

This package provides network modules that conform to UEFI 2.4 specification.

(C) Copyright 2015 Hewlett-Packard Development Company, L.P.

Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under
the terms and conditions of the BSD License which accompanies this distribution.
The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Package Basic Information

Version of DEC file specification	0x00010005
Package Guid	947988BE-8D5C-471a-893D-AD181C46BEBB

Known Issue

- OPTIONAL macro for function parameter can not be dealed with doxygen, so it disapear in this document!

INDEX

- PCD
- GUID
- Modules

Generated on Mon Sep 28 08:49:52 2015 for NetworkPkg[ALL] by 1.5.7.1

PCD

INDEX

- PcdsDynamic
- PcdsDynamicEx
- PcdsFeatureFlag
- PcdsFixedAtBuild
- PcdsPatchableInModule

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

PcdsDynamic

INDEX

- common

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo PCD &raquo PcdsDynamic

common

INDEX

- gEfiNetworkPkgTokenSpaceGuid.PcdDhcp6UidType

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

gEfiNetworkPkgTokenSpaceGuid.PcdDhcp6UidType

IPv6 DHCP Unique Identifier (DUID) Type configuration (From RFCs 3315 and 6355).
01 = DUID Based on Link-layer Address Plus Time [DUID-LLT]
04 = UUID-Based DHCPv6 Unique Identifier (DUID-UUID)
02 = DUID Assigned by Vendor Based on Enterprise Number [DUID-EN] (not supported)
03 = DUID Based on Link-layer Address [DUID-LL] (not supported)
Type Value of Dhcp6 Unique Identifier (DUID).

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdDhcp6UidType	gEfiNetworkPkgTokenSpaceGuid	0x10000001	UINT8	4

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

PcdsDynamicEx

INDEX

- common

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

common

INDEX

- gEfiNetworkPkgTokenSpaceGuid.PcdDhcp6UidType

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

gEfiNetworkPkgTokenSpaceGuid.PcdDhcp6UidType

IPv6 DHCP Unique Identifier (DUID) Type configuration (From RFCs 3315 and 6355).
01 = DUID Based on Link-layer Address Plus Time [DUID-LLT]
04 = UUID-Based DHCPv6 Unique Identifier (DUID-UUID)
02 = DUID Assigned by Vendor Based on Enterprise Number [DUID-EN] (not supported)
03 = DUID Based on Link-layer Address [DUID-LL] (not supported)
Type Value of Dhcp6 Unique Identifier (DUID).

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdDhcp6UidType	gEfiNetworkPkgTokenSpaceGuid	0x10000001	UINT8	4

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

PcdsFeatureFlag

INDEX

- common

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

common

INDEX

- gEfiNetworkPkgTokenSpaceGuid.PcdIpsecCertificateEnabled

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecCertificateE

Indicates if the IPsec IKEv2 Certificate Authentication feature is enabled or not.

TRUE - Certificate Authentication feature is enabled.

FALSE - Does not support Certificate Authentication.

Enable IPsec IKEv2 Certificate Authentication.

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdIpsecCertificateEnabled	gEfiNetworkPkgTokenSpaceGuid	0x00000007	BOOLEAN	TRUE

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

PcdsFixedAtBuild

INDEX

- common

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

common

INDEX

- gEfiNetworkPkgTokenSpaceGuid.PcdDhcp6UidType
- gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCaFile
- gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCaFileSize
- gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificate
- gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificateKey
- gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificateKeySize
- gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificateSize

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

gEfiNetworkPkgTokenSpaceGuid.PcdDhcp6UidType

IPv6 DHCP Unique Identifier (DUID) Type configuration (From RFCs 3315 and 6355).
01 = DUID Based on Link-layer Address Plus Time [DUID-LLT]
04 = UUID-Based DHCPv6 Unique Identifier (DUID-UUID)
02 = DUID Assigned by Vendor Based on Enterprise Number [DUID-EN] (not supported)
03 = DUID Based on Link-layer Address [DUID-LL] (not supported)
Type Value of Dhcp6 Unique Identifier (DUID).

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdDhcp6UidType	gEfiNetworkPkgTokenSpaceGuid	0x10000001	UINT8	4

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCaFile

CA certificate used by IPsec.
CA file.

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdIpsecUefiCaFile	gEfiNetworkPkgTokenSpaceGuid	0x00000001	VOID*	{0x30, 0x82, 0x02, 0x76, 0x30, 0x82, 0x01, 0xDF, 0xA0, 0x03, 0x02, 0x01, 0x02, 0x02, 0x09, 0x00, 0x80, 0x1D, 0xB9, 0x63, 0x93, 0x7C, 0x9D, 0xE0, 0x30, 0x0D, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x04, 0x05, 0x00, 0x30, 0x74, 0x31, 0x13, 0x30, 0x11, 0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x0A, 0x4D, 0x79, 0x20, 0x54, 0x65, 0x73, 0x74, 0x20, 0x43, 0x41, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x08, 0x13, 0x02, 0x48, 0x5A, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13, 0x02, 0x43, 0x4E, 0x31, 0x1C, 0x30, 0x1A, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x09, 0x01, 0x16, 0x0D, 0x74, 0x65, 0x73, 0x74, 0x40, 0x63, 0x65, 0x72, 0x74, 0x2E, 0x63, 0x6F, 0x6D, 0x31, 0x25, 0x30, 0x23, 0x06, 0x03, 0x55, 0x04, 0x0A, 0x13, 0x1C, 0x52, 0x6F, 0x6F, 0x74, 0x20, 0x43, 0x65, 0x72, 0x74, 0x69, 0x66, 0x69, 0x63, 0x61, 0x74, 0x69, 0x6F, 0x6E, 0x20, 0x41, 0x75, 0x74, 0x68, 0x6F, 0x72, 0x69, 0x74, 0x79, 0x30, 0x1E, 0x17, 0x0D, 0x31, 0x30, 0x31, 0x31, 0x30, 0x31, 0x30, 0x31, 0x35, 0x33, 0x33,

				0x37, 0x5A, 0x17, 0x0D, 0x31, 0x31, 0x31, 0x31, 0x30, 0x31, 0x30, 0x31, 0x35, 0x33, 0x33, 0x37, 0x5A, 0x30, 0x74, 0x31, 0x13, 0x30, 0x11, 0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x0A, 0x4D, 0x79, 0x20, 0x54, 0x65, 0x73, 0x74, 0x20, 0x43, 0x41, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x08, 0x13, 0x02, 0x48, 0x5A, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13, 0x02, 0x43, 0x4E, 0x31, 0x1C, 0x30, 0x1A, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x09, 0x01, 0x16, 0x0D, 0x74, 0x65, 0x73, 0x74, 0x40, 0x63, 0x65, 0x72, 0x74, 0x2E, 0x63, 0x6F, 0x6D, 0x31, 0x25, 0x30, 0x23, 0x06, 0x03, 0x55, 0x04, 0x0A, 0x13, 0x1C, 0x52, 0x6F, 0x6F, 0x74, 0x20, 0x43, 0x65, 0x72, 0x74, 0x69, 0x66, 0x69, 0x63, 0x61, 0x74, 0x69, 0x6F, 0x6E, 0x20, 0x41, 0x75, 0x74, 0x68, 0x6F, 0x72, 0x69, 0x74, 0x79, 0x30, 0x81, 0x9F, 0x30, 0x0D, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x01, 0x05, 0x00, 0x03, 0x81, 0x8D, 0x00, 0x30, 0x81, 0x89, 0x02, 0x81, 0x81, 0x00, 0xFC, 0x80, 0x5D, 0x32, 0x55, 0xC7, 0x4C, 0xC6, 0xA8, 0x2F, 0xF7, 0xEC, 0x1F, 0x75, 0x48, 0x02, 0x79, 0xEB, 0xDF, 0x17, 0x1B, 0x08, 0xBA, 0x21, 0xDD, 0xE5, 0x43, 0x06, 0xE8, 0x81, 0xC5, 0x50, 0x3C, 0x18, 0xDD, 0x53, 0xF4, 0xC9, 0xC9, 0xE1, 0x7A, 0xD3, 0xB3, 0x99, 0xA7, 0xC6, 0x43, 0x2A, 0x51, 0x65, 0x10, 0x93,

				0xBA, 0x5F, 0x48, 0xAC, 0x54, 0x12, 0x70, 0x9E, 0xF2, 0x9E, 0x7D, 0xF7, 0x22, 0xAA, 0xB7, 0x19, 0xDE, 0xA9, 0x4D, 0x55, 0xAA, 0x41, 0x8F, 0x08, 0xBD, 0x74, 0xFA, 0xE5, 0x57, 0x13, 0xB4, 0x30, 0x9A, 0xBA, 0x56, 0x01, 0x55, 0x8A, 0x9B, 0x5B, 0x50, 0x29, 0x82, 0xF9, 0x00, 0x69, 0x7E, 0x7B, 0x91, 0xA7, 0x2D, 0x48, 0x1A, 0x93, 0x7C, 0xA2, 0xF9, 0x06, 0x64, 0x4B, 0x80, 0xF8, 0x47, 0x58, 0x45, 0x90, 0x09, 0xEA, 0xD6, 0x7B, 0x85, 0x49, 0x2A, 0x4E, 0xB6, 0x71, 0x02, 0x03, 0x01, 0x00, 0x01, 0xA3, 0x10, 0x30, 0x0E, 0x30, 0x0C, 0x06, 0x03, 0x55, 0x1D, 0x13, 0x04, 0x05, 0x30, 0x03, 0x01, 0x01, 0xFF, 0x30, 0x0D, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x04, 0x05, 0x00, 0x03, 0x81, 0x81, 0x00, 0xEF, 0x38, 0x6A, 0x43, 0x1C, 0x1D, 0x37, 0xBD, 0xF7, 0xCF, 0x15, 0x6A, 0x99, 0x44, 0xE1, 0xFC, 0x68, 0x6E, 0x91, 0x31, 0x9C, 0x1E, 0x8C, 0x1F, 0x72, 0x4B, 0x93, 0x16, 0x1F, 0x06, 0xFE, 0x94, 0xA9, 0x41, 0x64, 0x81, 0xFD, 0xFF, 0xE7, 0x27, 0x4D, 0xE7, 0x59, 0x55, 0xE1, 0x20, 0x14, 0x07, 0x3C, 0x26, 0x78, 0xB0, 0x72, 0x48, 0x76, 0x0C, 0x8B, 0x3F, 0x08, 0xD0, 0x75, 0x7D, 0x76, 0xA4, 0xB5, 0x56, 0xA6, 0xC9, 0x88, 0x17, 0x27, 0x95, 0x85, 0xEE, 0x42, 0x1E, 0x15, 0x0B, 0x05, 0xDC, 0x2F, 0x97, 0x7B, 0x26, 0x82, 0x62, 0x23, 0xDF, 0xBF, 0x55, 0x09, 0xBF, 0x5E, 0x28, 0x1A, 0xCA, 0x1B,

				0xEC, 0xA4, 0x81, 0xB7, 0x9D, 0x91, 0xC9, 0x60, 0x5B, 0x29, 0x2B, 0x4C, 0x6F, 0x8B, 0xCC, 0x17, 0xA8, 0xD6, 0x5D, 0x6B, 0xBC, 0x0D, 0x03, 0x31, 0xB0, 0x57, 0xC9, 0xF8, 0x59, 0x88, 0x3D}

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo PCD &raquo PcdsFixedAtBuild &raquo common

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCaFileSi

CA certificate file's size.
CA file's size.

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdIpsecUefiCaFileSize	gEfiNetworkPkgTokenSpaceGuid	0x00000002	UINT32	0x0000027A

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertifica

X509 certificate as Public Key which is used by IPsec (DER format)
Pubic Key for remote peer.

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdIpsecUefiCertificate	gEfiNetworkPkgTokenSpaceGuid	0x00000003	VOID*	{0x30, 0x82, 0x02, 0x4D, 0x30, 0x82, 0x01, 0xB6, 0x02, 0x01, 0x01, 0x30, 0x0D, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x04, 0x05, 0x00, 0x30, 0x74, 0x31, 0x13, 0x30, 0x11, 0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x0A, 0x4D, 0x79, 0x20, 0x54, 0x65, 0x73, 0x74, 0x20, 0x43, 0x41, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x08, 0x13, 0x02, 0x48, 0x5A, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13, 0x02, 0x43, 0x4E, 0x31, 0x1C, 0x30, 0x1A, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x09, 0x01, 0x16, 0x0D, 0x74, 0x65, 0x73, 0x74, 0x40, 0x63, 0x65, 0x72, 0x74, 0x2E, 0x63, 0x6F, 0x6D, 0x31, 0x25, 0x30, 0x23, 0x06, 0x03, 0x55, 0x04, 0x0A, 0x13, 0x1C, 0x52, 0x6F, 0x6F, 0x74, 0x20, 0x43, 0x65, 0x72, 0x74, 0x69, 0x66, 0x69, 0x63,

				0x61, 0x74, 0x69, 0x6F, 0x6E, 0x20, 0x41, 0x75, 0x74, 0x68, 0x6F, 0x72, 0x69, 0x74, 0x79, 0x30, 0x1E, 0x17, 0x0D, 0x31, 0x30, 0x31, 0x31, 0x30, 0x31, 0x30, 0x32, 0x30, 0x34, 0x35, 0x39, 0x5A, 0x17, 0x0D, 0x31, 0x31, 0x31, 0x31, 0x30, 0x31, 0x30, 0x32, 0x30, 0x34, 0x35, 0x39, 0x5A, 0x30, 0x6A, 0x31, 0x0D, 0x30, 0x0B, 0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x04, 0x55, 0x45, 0x46, 0x49, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x08, 0x13, 0x02, 0x53, 0x48, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13, 0x02, 0x43, 0x4E, 0x31, 0x23, 0x30, 0x21, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x09, 0x01, 0x16, 0x14, 0x75, 0x65, 0x66, 0x69, 0x2E, 0x74, 0x69, 0x61, 0x6E, 0x6F, 0x40, 0x69, 0x6E, 0x74, 0x65, 0x6C, 0x2E, 0x63, 0x6F, 0x6D, 0x31, 0x0C, 0x30, 0x0A, 0x06, 0x03, 0x55, 0x04, 0x0B, 0x13, 0x03, 0x53, 0x53, 0x47, 0x31, 0x0C, 0x30, 0x0A, 0x06, 0x03, 0x55, 0x04, 0x0A, 0x13, 0x03, 0x53, 0x53, 0x47, 0x30, 0x81, 0x9F, 0x30, 0x0D, 0x06, 0x09,

				0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x01, 0x05, 0x00, 0x03, 0x81, 0x8D, 0x00, 0x30, 0x81, 0x89, 0x02, 0x81, 0x81, 0x00, 0xE9, 0x90, 0x47, 0x0D, 0x79, 0x93, 0xED, 0xF5, 0xBD, 0xC9, 0x56, 0x03, 0xDF, 0xE2, 0x71, 0xA9, 0x42, 0x3B, 0x20, 0x1E, 0xAF, 0x88, 0x9D, 0x3F, 0xE1, 0xDE, 0x61, 0xEE, 0x83, 0xC4, 0x2E, 0x48, 0x7A, 0x1F, 0x86, 0x54, 0xD2, 0xD5, 0x61, 0x94, 0xE1, 0x15, 0x79, 0x65, 0xCB, 0x39, 0xEE, 0x78, 0x68, 0x3D, 0x2C, 0xEB, 0xE4, 0x7A, 0x8D, 0x98, 0x14, 0x28, 0x7E, 0x6B, 0xFD, 0xC5, 0xF5, 0x1B, 0x62, 0xB9, 0x86, 0x7C, 0xA1, 0x7C, 0xE9, 0x8F, 0xC8, 0xF4, 0xF3, 0x95, 0x5A, 0xAF, 0x0C, 0x21, 0x39, 0xEA, 0x47, 0x5A, 0x1E, 0xBD, 0xBE, 0x7F, 0x1B, 0x0F, 0x31, 0xFB, 0xBD, 0x57, 0xAE, 0xD7, 0xCB, 0x46, 0x83, 0x8B, 0x16, 0x19, 0x74, 0xD9, 0x9E, 0x2D, 0x18, 0xE6, 0xA4, 0x5F, 0x90, 0x90, 0x54, 0xE1, 0x4B, 0x7B, 0x57, 0x76, 0xBD, 0xF4, 0xC0, 0x4D, 0x79, 0x5F, 0x64, 0x6C, 0x0D, 0x2D, 0x02, 0x03, 0x01, 0x00, 0x01, 0x30, 0x0D, 0x06, 0x09, 0x2A, 0x86, 0x48,

				0x86, 0xF7, 0x0D, 0x01, 0x01, 0x04, 0x05, 0x00, 0x03, 0x81, 0x81, 0x00, 0x5A, 0x80, 0x5F, 0xD3, 0x3C, 0x93, 0x81, 0xB9, 0x1B, 0xAA, 0x08, 0x1F, 0x47, 0x9C, 0x88, 0xF3, 0x1E, 0xE6, 0x6B, 0xBB, 0x99, 0xE6, 0x23, 0x1A, 0xCB, 0x25, 0x81, 0x54, 0x51, 0x88, 0xDF, 0x9B, 0xC6, 0xBF, 0x60, 0xDB, 0x6C, 0x5D, 0x69, 0xB1, 0x3A, 0xDE, 0x94, 0xEE, 0xD7, 0x6C, 0xF2, 0x2D, 0x63, 0xD3, 0xB3, 0xAB, 0xE6, 0xB5, 0x0A, 0xBF, 0xCE, 0x61, 0xC0, 0xD3, 0x73, 0x9E, 0x80, 0xB5, 0x0C, 0xC0, 0x03, 0x57, 0xA9, 0x56, 0x59, 0x1B, 0xA2, 0x99, 0x03, 0xA6, 0xA3, 0xC4, 0x59, 0xB3, 0xD9, 0x14, 0xA1, 0x34, 0x18, 0xF3, 0x73, 0xB8, 0x54, 0xAA, 0xED, 0x7D, 0x31, 0x3E, 0x23, 0xAD, 0xF1, 0x86, 0xF7, 0xE6, 0xD9, 0x01, 0x0D, 0x68, 0xC6, 0xC5, 0x95, 0x18, 0xD2, 0x89, 0xB7, 0x06, 0x96, 0xC9, 0x11, 0xB9, 0xF0, 0xDA, 0xD9, 0x02, 0x25, 0xC4, 0xB9, 0x72, 0xF8, 0x6D, 0xC5, 0x5B}

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertifica

Private Key used by IPsec (PEM format).
Private Key.

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdIpsecUefiCertificateKey	gEfiNetworkPkgTokenSpaceGuid	0x00000005	VOID*	{0x2D, 0x2D, 0x2D, 0x2D, 0x2D, 0x42, 0x45, 0x47, 0x49, 0x4E, 0x20, 0x52, 0x53, 0x41, 0x20, 0x50, 0x52, 0x49, 0x56, 0x41, 0x54, 0x45, 0x20, 0x4B, 0x45, 0x59, 0x2D, 0x2D, 0x2D, 0x2D, 0x2D, 0x2D, 0x0A, 0x50, 0x72, 0x6F, 0x63, 0x2D, 0x54, 0x79, 0x70, 0x65, 0x3A, 0x20, 0x34, 0x2C, 0x45, 0x4E, 0x43, 0x52, 0x59, 0x50, 0x54, 0x45, 0x44, 0x0A, 0x44, 0x45, 0x4B, 0x2D, 0x49, 0x6E, 0x66, 0x6F, 0x3A, 0x20, 0x44, 0x45, 0x53, 0x2D, 0x45, 0x44, 0x45, 0x33, 0x2D, 0x43, 0x42, 0x43, 0x2C, 0x32, 0x42, 0x31, 0x46, 0x42, 0x41, 0x43, 0x41, 0x38, 0x36, 0x32, 0x36, 0x33, 0x34, 0x41, 0x37, 0x0A, 0x0A, 0x61, 0x52, 0x78, 0x49, 0x58, 0x33, 0x59, 0x4D, 0x68, 0x49, 0x50, 0x41, 0x73, 0x59, 0x79, 0x6F, 0x6A, 0x49, 0x76, 0x46, 0x7A, 0x42, 0x75,

				0x6B, 0x74, 0x6B, 0x4A, 0x47, 0x5A, 0x38, 0x4D, 0x64, 0x33, 0x5A, 0x53, 0x73, 0x39, 0x41, 0x2B, 0x52, 0x2B, 0x57, 0x45, 0x59, 0x41, 0x70, 0x34, 0x63, 0x4F, 0x55, 0x43, 0x4A, 0x78, 0x51, 0x2F, 0x66, 0x4A, 0x38, 0x58, 0x4F, 0x45, 0x64, 0x58, 0x38, 0x0A, 0x31, 0x63, 0x4E, 0x66, 0x4B, 0x2B, 0x49, 0x62, 0x76, 0x4B, 0x4D, 0x68, 0x55, 0x67, 0x30, 0x4B, 0x4E, 0x35, 0x38, 0x37, 0x71, 0x66, 0x2F, 0x4C, 0x31, 0x76, 0x57, 0x58, 0x6F, 0x31, 0x74, 0x5A, 0x6B, 0x59, 0x2B, 0x5A, 0x53, 0x4E, 0x63, 0x46, 0x45, 0x41, 0x76, 0x37, 0x43, 0x43, 0x50, 0x51, 0x6B, 0x64, 0x4A, 0x42, 0x48, 0x35, 0x65, 0x6B, 0x35, 0x44, 0x51, 0x2F, 0x37, 0x6D, 0x71, 0x55, 0x0A, 0x6B, 0x76, 0x78, 0x48, 0x53, 0x50, 0x70, 0x34, 0x66, 0x41, 0x71, 0x47, 0x61, 0x68, 0x54, 0x31, 0x75, 0x37, 0x37, 0x56, 0x66, 0x4E, 0x66, 0x31, 0x53, 0x74, 0x61, 0x73, 0x31, 0x6E, 0x4F, 0x67, 0x6A, 0x50, 0x31, 0x41, 0x6C, 0x7A, 0x6E, 0x6B, 0x6A, 0x57, 0x61, 0x72,

				0x6A, 0x51, 0x4F, 0x73, 0x48, 0x46, 0x33, 0x41, 0x46, 0x31, 0x62, 0x61, 0x51, 0x4A, 0x50, 0x5A, 0x31, 0x6A, 0x71, 0x4C, 0x0A, 0x61, 0x30, 0x49, 0x45, 0x6E, 0x30, 0x6C, 0x59, 0x6C, 0x78, 0x35, 0x79, 0x4D, 0x6D, 0x78, 0x54, 0x47, 0x57, 0x79, 0x52, 0x35, 0x70, 0x57, 0x51, 0x35, 0x71, 0x66, 0x78, 0x2B, 0x62, 0x37, 0x64, 0x37, 0x75, 0x71, 0x67, 0x47, 0x69, 0x66, 0x36, 0x6A, 0x44, 0x47, 0x4D, 0x37, 0x68, 0x38, 0x43, 0x78, 0x2F, 0x74, 0x67, 0x2B, 0x61, 0x62, 0x45, 0x31, 0x34, 0x30, 0x2F, 0x50, 0x66, 0x6C, 0x33, 0x0A, 0x33, 0x6A, 0x50, 0x6C, 0x52, 0x75, 0x73, 0x57, 0x6F, 0x6F, 0x63, 0x49, 0x41, 0x76, 0x49, 0x74, 0x79, 0x51, 0x6D, 0x39, 0x39, 0x71, 0x74, 0x34, 0x64, 0x6E, 0x74, 0x6E, 0x74, 0x6F, 0x4A, 0x43, 0x6D, 0x4F, 0x53, 0x79, 0x71, 0x67, 0x4D, 0x6E, 0x76, 0x2F, 0x76, 0x2B, 0x51, 0x48, 0x74, 0x79, 0x4D, 0x73, 0x42, 0x64, 0x38, 0x34, 0x78, 0x45, 0x57, 0x46, 0x36, 0x72, 0x58, 0x4D, 0x52, 0x63, 0x0A, 0x53,

				0x2B, 0x66, 0x68, 0x54, 0x71, 0x58, 0x74, 0x54, 0x38, 0x44, 0x50, 0x65, 0x70, 0x2F, 0x56, 0x44, 0x66, 0x65, 0x78, 0x6B, 0x41, 0x63, 0x6D, 0x63, 0x75, 0x41, 0x69, 0x6F, 0x2B, 0x79, 0x64, 0x51, 0x75, 0x49, 0x31, 0x32, 0x7A, 0x50, 0x70, 0x45, 0x68, 0x50, 0x45, 0x68, 0x31, 0x44, 0x50, 0x58, 0x73, 0x64, 0x58, 0x67, 0x64, 0x77, 0x39, 0x75, 0x46, 0x47, 0x6D, 0x63, 0x35, 0x68, 0x52, 0x0A, 0x35, 0x31, 0x57, 0x41, 0x31, 0x65, 0x63, 0x44, 0x48, 0x6A, 0x31, 0x58, 0x32, 0x45, 0x72, 0x36, 0x39, 0x59, 0x70, 0x31, 0x50, 0x69, 0x43, 0x37, 0x49, 0x47, 0x79, 0x6F, 0x71, 0x57, 0x43, 0x37, 0x69, 0x2F, 0x71, 0x6D, 0x6D, 0x72, 0x49, 0x66, 0x6F, 0x41, 0x54, 0x74, 0x39, 0x58, 0x34, 0x30, 0x54, 0x56, 0x63, 0x37, 0x42, 0x63, 0x6A, 0x34, 0x63, 0x54, 0x31, 0x78, 0x37, 0x6B, 0x70, 0x4F, 0x0A, 0x4C, 0x71, 0x67, 0x33, 0x6C, 0x50, 0x78, 0x33, 0x2B, 0x4A, 0x63, 0x33, 0x43, 0x67, 0x34, 0x79, 0x5A, 0x54, 0x66, 0x6E, 0x4A, 0x5A, 0x37, 0x48, 0x76, 0x36, 0x64, 0x68,

				0x67, 0x45, 0x6D, 0x70, 0x4D, 0x73, 0x74, 0x46, 0x65, 0x35, 0x34, 0x49, 0x53, 0x76, 0x74, 0x38, 0x37, 0x59, 0x4E, 0x77, 0x74, 0x4C, 0x65, 0x6C, 0x34, 0x67, 0x50, 0x4A, 0x79, 0x53, 0x42, 0x30, 0x4B, 0x76, 0x37, 0x69, 0x0A, 0x33, 0x32, 0x74, 0x37, 0x67, 0x4F, 0x30, 0x79, 0x6D, 0x73, 0x62, 0x71, 0x4A, 0x55, 0x75, 0x79, 0x41, 0x68, 0x47, 0x64, 0x33, 0x63, 0x2B, 0x78, 0x4C, 0x46, 0x2F, 0x63, 0x63, 0x4F, 0x57, 0x44, 0x52, 0x34, 0x79, 0x72, 0x30, 0x6A, 0x79, 0x64, 0x74, 0x70, 0x79, 0x69, 0x64, 0x52, 0x45, 0x66, 0x56, 0x46, 0x66, 0x53, 0x6C, 0x39, 0x54, 0x30, 0x6D, 0x53, 0x72, 0x4E, 0x76, 0x43, 0x71, 0x45, 0x0A, 0x52, 0x52, 0x5A, 0x6E, 0x42, 0x56, 0x76, 0x37, 0x50, 0x66, 0x6C, 0x75, 0x72, 0x31, 0x59, 0x35, 0x70, 0x2F, 0x65, 0x78, 0x54, 0x63, 0x56, 0x34, 0x72, 0x4B, 0x52, 0x69, 0x6C, 0x35, 0x58, 0x6A, 0x2F, 0x39, 0x59, 0x56, 0x31, 0x4E, 0x6E, 0x6D, 0x4E, 0x2B, 0x2F, 0x31, 0x31, 0x74, 0x36, 0x58, 0x74, 0x6A, 0x72, 0x75, 0x52, 0x62,

				0x33, 0x79, 0x70, 0x38, 0x76, 0x64, 0x6C, 0x61, 0x65, 0x5A, 0x0A, 0x6C, 0x67, 0x45, 0x69, 0x73, 0x30, 0x42, 0x7A, 0x4B, 0x59, 0x39, 0x59, 0x64, 0x58, 0x48, 0x64, 0x46, 0x58, 0x57, 0x59, 0x4F, 0x41, 0x71, 0x50, 0x48, 0x45, 0x65, 0x4B, 0x57, 0x79, 0x61, 0x59, 0x5A, 0x56, 0x79, 0x43, 0x70, 0x51, 0x65, 0x43, 0x53, 0x71, 0x4F, 0x71, 0x48, 0x38, 0x67, 0x42, 0x6B, 0x4F, 0x62, 0x43, 0x69, 0x72, 0x41, 0x6A, 0x65, 0x56, 0x70, 0x35, 0x7A, 0x37, 0x6B, 0x31, 0x0A, 0x64, 0x4F, 0x2F, 0x6D, 0x56, 0x74, 0x49, 0x2B, 0x57, 0x47, 0x30, 0x48, 0x72, 0x37, 0x5A, 0x4C, 0x53, 0x52, 0x78, 0x6F, 0x61, 0x44, 0x47, 0x42, 0x33, 0x4E, 0x35, 0x38, 0x4B, 0x56, 0x45, 0x4F, 0x34, 0x65, 0x46, 0x56, 0x75, 0x6E, 0x59, 0x77, 0x51, 0x42, 0x54, 0x7A, 0x4F, 0x65, 0x57, 0x39, 0x6C, 0x4B, 0x79, 0x49, 0x38, 0x67, 0x4D, 0x45, 0x57, 0x6C, 0x62, 0x4B, 0x72, 0x41, 0x45, 0x49, 0x0A, 0x46, 0x4B, 0x38, 0x7A, 0x58, 0x6F, 0x44, 0x74, 0x39, 0x6A,

				0x7A, 0x54, 0x37, 0x67, 0x68, 0x6A, 0x79, 0x45, 0x54, 0x67, 0x44, 0x6C, 0x69, 0x50, 0x53, 0x49, 0x46, 0x6A, 0x79, 0x31, 0x64, 0x6B, 0x6A, 0x6D, 0x68, 0x53, 0x78, 0x79, 0x6A, 0x67, 0x62, 0x71, 0x45, 0x3D, 0x0A, 0x2D, 0x2D, 0x2D, 0x2D, 0x2D, 0x45, 0x4E, 0x44, 0x20, 0x52, 0x53, 0x41, 0x20, 0x50, 0x52, 0x49, 0x56, 0x41, 0x54, 0x45, 0x20, 0x4B, 0x45, 0x59, 0x2D, 0x2D, 0x2D, 0x2D, 0x2D, 0x2D, 0x0A}

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo PCD &raquo PcdsFixedAtBuild &raquo common

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertifica

Private Key's size.
Private Key's size.

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdIpsecUefiCertificateKeySize	gEfiNetworkPkgTokenSpaceGuid	0x00000006	UINT32	0x3d5

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertifica

X509 certificate as Public Key's size.
Pubic Key's size.

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdIpsecUefiCertificateSize	gEfiNetworkPkgTokenSpaceGuid	0x00000004	UINT32	0x251

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

PcdsPatchableInModule

INDEX

- common

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

common

INDEX

- gEfiNetworkPkgTokenSpaceGuid.PcdDhcp6UidType
- gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCaFile
- gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCaFileSize
- gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificate
- gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificateKey
- gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificateKeySize
- gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertificateSize

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

gEfiNetworkPkgTokenSpaceGuid.PcdDhcp6UidType

IPv6 DHCP Unique Identifier (DUID) Type configuration (From RFCs 3315 and 6355).
01 = DUID Based on Link-layer Address Plus Time [DUID-LLT]
04 = UUID-Based DHCPv6 Unique Identifier (DUID-UUID)
02 = DUID Assigned by Vendor Based on Enterprise Number [DUID-EN] (not supported)
03 = DUID Based on Link-layer Address [DUID-LL] (not supported)
Type Value of Dhcp6 Unique Identifier (DUID).

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdDhcp6UidType	gEfiNetworkPkgTokenSpaceGuid	0x10000001	UINT8	4

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCaFile

CA certificate used by IPsec.
CA file.

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdIpsecUefiCaFile	gEfiNetworkPkgTokenSpaceGuid	0x00000001	VOID*	{0x30, 0x82, 0x02, 0x76, 0x30, 0x82, 0x01, 0xDF, 0xA0, 0x03, 0x02, 0x01, 0x02, 0x02, 0x09, 0x00, 0x80, 0x1D, 0xB9, 0x63, 0x93, 0x7C, 0x9D, 0xE0, 0x30, 0x0D, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x04, 0x05, 0x00, 0x30, 0x74, 0x31, 0x13, 0x30, 0x11, 0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x0A, 0x4D, 0x79, 0x20, 0x54, 0x65, 0x73, 0x74, 0x20, 0x43, 0x41, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x08, 0x13, 0x02, 0x48, 0x5A, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13, 0x02, 0x43, 0x4E, 0x31, 0x1C, 0x30, 0x1A, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x09, 0x01, 0x16, 0x0D, 0x74, 0x65, 0x73, 0x74, 0x40, 0x63, 0x65, 0x72, 0x74, 0x2E, 0x63, 0x6F, 0x6D, 0x31, 0x25, 0x30, 0x23, 0x06, 0x03, 0x55, 0x04, 0x0A, 0x13, 0x1C, 0x52, 0x6F, 0x6F, 0x74, 0x20, 0x43, 0x65, 0x72, 0x74, 0x69, 0x66, 0x69, 0x63, 0x61, 0x74, 0x69, 0x6F, 0x6E, 0x20, 0x41, 0x75, 0x74, 0x68, 0x6F, 0x72, 0x69, 0x74, 0x79, 0x30, 0x1E, 0x17, 0x0D, 0x31, 0x30, 0x31, 0x31, 0x30, 0x31, 0x30, 0x31, 0x35, 0x33, 0x33,

				0x37, 0x5A, 0x17, 0x0D, 0x31, 0x31, 0x31, 0x31, 0x30, 0x31, 0x30, 0x31, 0x35, 0x33, 0x33, 0x37, 0x5A, 0x30, 0x74, 0x31, 0x13, 0x30, 0x11, 0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x0A, 0x4D, 0x79, 0x20, 0x54, 0x65, 0x73, 0x74, 0x20, 0x43, 0x41, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x08, 0x13, 0x02, 0x48, 0x5A, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13, 0x02, 0x43, 0x4E, 0x31, 0x1C, 0x30, 0x1A, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x09, 0x01, 0x16, 0x0D, 0x74, 0x65, 0x73, 0x74, 0x40, 0x63, 0x65, 0x72, 0x74, 0x2E, 0x63, 0x6F, 0x6D, 0x31, 0x25, 0x30, 0x23, 0x06, 0x03, 0x55, 0x04, 0x0A, 0x13, 0x1C, 0x52, 0x6F, 0x6F, 0x74, 0x20, 0x43, 0x65, 0x72, 0x74, 0x69, 0x66, 0x69, 0x63, 0x61, 0x74, 0x69, 0x6F, 0x6E, 0x20, 0x41, 0x75, 0x74, 0x68, 0x6F, 0x72, 0x69, 0x74, 0x79, 0x30, 0x81, 0x9F, 0x30, 0x0D, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x01, 0x05, 0x00, 0x03, 0x81, 0x8D, 0x00, 0x30, 0x81, 0x89, 0x02, 0x81, 0x81, 0x00, 0xFC, 0x80, 0x5D, 0x32, 0x55, 0xC7, 0x4C, 0xC6, 0xA8, 0x2F, 0xF7, 0xEC, 0x1F, 0x75, 0x48, 0x02, 0x79, 0xEB, 0xDF, 0x17, 0x1B, 0x08, 0xBA, 0x21, 0xDD, 0xE5, 0x43, 0x06, 0xE8, 0x81, 0xC5, 0x50, 0x3C, 0x18, 0xDD, 0x53, 0xF4, 0xC9, 0xC9, 0xE1, 0x7A, 0xD3, 0xB3, 0x99, 0xA7, 0xC6, 0x43, 0x2A, 0x51, 0x65, 0x10, 0x93,

				0xBA, 0x5F, 0x48, 0xAC, 0x54, 0x12, 0x70, 0x9E, 0xF2, 0x9E, 0x7D, 0xF7, 0x22, 0xAA, 0xB7, 0x19, 0xDE, 0xA9, 0x4D, 0x55, 0xAA, 0x41, 0x8F, 0x08, 0xBD, 0x74, 0xFA, 0xE5, 0x57, 0x13, 0xB4, 0x30, 0x9A, 0xBA, 0x56, 0x01, 0x55, 0x8A, 0x9B, 0x5B, 0x50, 0x29, 0x82, 0xF9, 0x00, 0x69, 0x7E, 0x7B, 0x91, 0xA7, 0x2D, 0x48, 0x1A, 0x93, 0x7C, 0xA2, 0xF9, 0x06, 0x64, 0x4B, 0x80, 0xF8, 0x47, 0x58, 0x45, 0x90, 0x09, 0xEA, 0xD6, 0x7B, 0x85, 0x49, 0x2A, 0x4E, 0xB6, 0x71, 0x02, 0x03, 0x01, 0x00, 0x01, 0xA3, 0x10, 0x30, 0x0E, 0x30, 0x0C, 0x06, 0x03, 0x55, 0x1D, 0x13, 0x04, 0x05, 0x30, 0x03, 0x01, 0x01, 0xFF, 0x30, 0x0D, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x04, 0x05, 0x00, 0x03, 0x81, 0x81, 0x00, 0xEF, 0x38, 0x6A, 0x43, 0x1C, 0x1D, 0x37, 0xBD, 0xF7, 0xCF, 0x15, 0x6A, 0x99, 0x44, 0xE1, 0xFC, 0x68, 0x6E, 0x91, 0x31, 0x9C, 0x1E, 0x8C, 0x1F, 0x72, 0x4B, 0x93, 0x16, 0x1F, 0x06, 0xFE, 0x94, 0xA9, 0x41, 0x64, 0x81, 0xFD, 0xFF, 0xE7, 0x27, 0x4D, 0xE7, 0x59, 0x55, 0xE1, 0x20, 0x14, 0x07, 0x3C, 0x26, 0x78, 0xB0, 0x72, 0x48, 0x76, 0x0C, 0x8B, 0x3F, 0x08, 0xD0, 0x75, 0x7D, 0x76, 0xA4, 0xB5, 0x56, 0xA6, 0xC9, 0x88, 0x17, 0x27, 0x95, 0x85, 0xEE, 0x42, 0x1E, 0x15, 0x0B, 0x05, 0xDC, 0x2F, 0x97, 0x7B, 0x26, 0x82, 0x62, 0x23, 0xDF, 0xBF, 0x55, 0x09, 0xBF, 0x5E, 0x28, 0x1A, 0xCA, 0x1B,

				0xEC, 0xA4, 0x81, 0xB7, 0x9D, 0x91, 0xC9, 0x60, 0x5B, 0x29, 0x2B, 0x4C, 0x6F, 0x8B, 0xCC, 0x17, 0xA8, 0xD6, 0x5D, 0x6B, 0xBC, 0x0D, 0x03, 0x31, 0xB0, 0x57, 0xC9, 0xF8, 0x59, 0x88, 0x3D}

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo PCD &raquo PcdsPatchableInModule &raquo common

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCaFileSi

CA certificate file's size.
CA file's size.

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdIpsecUefiCaFileSize	gEfiNetworkPkgTokenSpaceGuid	0x00000002	UINT32	0x0000027A

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertifica

X509 certificate as Public Key which is used by IPsec (DER format)
Pubic Key for remote peer.

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdIpsecUefiCertificate	gEfiNetworkPkgTokenSpaceGuid	0x00000003	VOID*	{0x30, 0x82, 0x02, 0x4D, 0x30, 0x82, 0x01, 0xB6, 0x02, 0x01, 0x01, 0x30, 0x0D, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x04, 0x05, 0x00, 0x30, 0x74, 0x31, 0x13, 0x30, 0x11, 0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x0A, 0x4D, 0x79, 0x20, 0x54, 0x65, 0x73, 0x74, 0x20, 0x43, 0x41, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x08, 0x13, 0x02, 0x48, 0x5A, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13, 0x02, 0x43, 0x4E, 0x31, 0x1C, 0x30, 0x1A, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x09, 0x01, 0x16, 0x0D, 0x74, 0x65, 0x73, 0x74, 0x40, 0x63, 0x65, 0x72, 0x74, 0x2E, 0x63, 0x6F, 0x6D, 0x31, 0x25, 0x30, 0x23, 0x06, 0x03, 0x55, 0x04, 0x0A, 0x13, 0x1C, 0x52, 0x6F, 0x6F, 0x74, 0x20, 0x43, 0x65, 0x72, 0x74, 0x69, 0x66, 0x69, 0x63,

				0x61, 0x74, 0x69, 0x6F, 0x6E, 0x20, 0x41, 0x75, 0x74, 0x68, 0x6F, 0x72, 0x69, 0x74, 0x79, 0x30, 0x1E, 0x17, 0x0D, 0x31, 0x30, 0x31, 0x31, 0x30, 0x31, 0x30, 0x32, 0x30, 0x34, 0x35, 0x39, 0x5A, 0x17, 0x0D, 0x31, 0x31, 0x31, 0x31, 0x30, 0x31, 0x30, 0x32, 0x30, 0x34, 0x35, 0x39, 0x5A, 0x30, 0x6A, 0x31, 0x0D, 0x30, 0x0B, 0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x04, 0x55, 0x45, 0x46, 0x49, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x08, 0x13, 0x02, 0x53, 0x48, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13, 0x02, 0x43, 0x4E, 0x31, 0x23, 0x30, 0x21, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x09, 0x01, 0x16, 0x14, 0x75, 0x65, 0x66, 0x69, 0x2E, 0x74, 0x69, 0x61, 0x6E, 0x6F, 0x40, 0x69, 0x6E, 0x74, 0x65, 0x6C, 0x2E, 0x63, 0x6F, 0x6D, 0x31, 0x0C, 0x30, 0x0A, 0x06, 0x03, 0x55, 0x04, 0x0B, 0x13, 0x03, 0x53, 0x53, 0x47, 0x31, 0x0C, 0x30, 0x0A, 0x06, 0x03, 0x55, 0x04, 0x0A, 0x13, 0x03, 0x53, 0x53, 0x47, 0x30, 0x81, 0x9F, 0x30, 0x0D, 0x06, 0x09,

				0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x01, 0x05, 0x00, 0x03, 0x81, 0x8D, 0x00, 0x30, 0x81, 0x89, 0x02, 0x81, 0x81, 0x00, 0xE9, 0x90, 0x47, 0x0D, 0x79, 0x93, 0xED, 0xF5, 0xBD, 0xC9, 0x56, 0x03, 0xDF, 0xE2, 0x71, 0xA9, 0x42, 0x3B, 0x20, 0x1E, 0xAF, 0x88, 0x9D, 0x3F, 0xE1, 0xDE, 0x61, 0xEE, 0x83, 0xC4, 0x2E, 0x48, 0x7A, 0x1F, 0x86, 0x54, 0xD2, 0xD5, 0x61, 0x94, 0xE1, 0x15, 0x79, 0x65, 0xCB, 0x39, 0xEE, 0x78, 0x68, 0x3D, 0x2C, 0xEB, 0xE4, 0x7A, 0x8D, 0x98, 0x14, 0x28, 0x7E, 0x6B, 0xFD, 0xC5, 0xF5, 0x1B, 0x62, 0xB9, 0x86, 0x7C, 0xA1, 0x7C, 0xE9, 0x8F, 0xC8, 0xF4, 0xF3, 0x95, 0x5A, 0xAF, 0x0C, 0x21, 0x39, 0xEA, 0x47, 0x5A, 0x1E, 0xBD, 0xBE, 0x7F, 0x1B, 0x0F, 0x31, 0xFB, 0xBD, 0x57, 0xAE, 0xD7, 0xCB, 0x46, 0x83, 0x8B, 0x16, 0x19, 0x74, 0xD9, 0x9E, 0x2D, 0x18, 0xE6, 0xA4, 0x5F, 0x90, 0x90, 0x54, 0xE1, 0x4B, 0x7B, 0x57, 0x76, 0xBD, 0xF4, 0xC0, 0x4D, 0x79, 0x5F, 0x64, 0x6C, 0x0D, 0x2D, 0x02, 0x03, 0x01, 0x00, 0x01, 0x30, 0x0D, 0x06, 0x09, 0x2A, 0x86, 0x48,

				0x86, 0xF7, 0x0D, 0x01, 0x01, 0x04, 0x05, 0x00, 0x03, 0x81, 0x81, 0x00, 0x5A, 0x80, 0x5F, 0xD3, 0x3C, 0x93, 0x81, 0xB9, 0x1B, 0xAA, 0x08, 0x1F, 0x47, 0x9C, 0x88, 0xF3, 0x1E, 0xE6, 0x6B, 0xBB, 0x99, 0xE6, 0x23, 0x1A, 0xCB, 0x25, 0x81, 0x54, 0x51, 0x88, 0xDF, 0x9B, 0xC6, 0xBF, 0x60, 0xDB, 0x6C, 0x5D, 0x69, 0xB1, 0x3A, 0xDE, 0x94, 0xEE, 0xD7, 0x6C, 0xF2, 0x2D, 0x63, 0xD3, 0xB3, 0xAB, 0xE6, 0xB5, 0x0A, 0xBF, 0xCE, 0x61, 0xC0, 0xD3, 0x73, 0x9E, 0x80, 0xB5, 0x0C, 0xC0, 0x03, 0x57, 0xA9, 0x56, 0x59, 0x1B, 0xA2, 0x99, 0x03, 0xA6, 0xA3, 0xC4, 0x59, 0xB3, 0xD9, 0x14, 0xA1, 0x34, 0x18, 0xF3, 0x73, 0xB8, 0x54, 0xAA, 0xED, 0x7D, 0x31, 0x3E, 0x23, 0xAD, 0xF1, 0x86, 0xF7, 0xE6, 0xD9, 0x01, 0x0D, 0x68, 0xC6, 0xC5, 0x95, 0x18, 0xD2, 0x89, 0xB7, 0x06, 0x96, 0xC9, 0x11, 0xB9, 0xF0, 0xDA, 0xD9, 0x02, 0x25, 0xC4, 0xB9, 0x72, 0xF8, 0x6D, 0xC5, 0x5B}

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertifica

Private Key used by IPsec (PEM format).
Private Key.

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdIpsecUefiCertificateKey	gEfiNetworkPkgTokenSpaceGuid	0x00000005	VOID*	{0x2D, 0x2D, 0x2D, 0x2D, 0x2D, 0x42, 0x45, 0x47, 0x49, 0x4E, 0x20, 0x52, 0x53, 0x41, 0x20, 0x50, 0x52, 0x49, 0x56, 0x41, 0x54, 0x45, 0x20, 0x4B, 0x45, 0x59, 0x2D, 0x2D, 0x2D, 0x2D, 0x2D, 0x2D, 0x0A, 0x50, 0x72, 0x6F, 0x63, 0x2D, 0x54, 0x79, 0x70, 0x65, 0x3A, 0x20, 0x34, 0x2C, 0x45, 0x4E, 0x43, 0x52, 0x59, 0x50, 0x54, 0x45, 0x44, 0x0A, 0x44, 0x45, 0x4B, 0x2D, 0x49, 0x6E, 0x66, 0x6F, 0x3A, 0x20, 0x44, 0x45, 0x53, 0x2D, 0x45, 0x44, 0x45, 0x33, 0x2D, 0x43, 0x42, 0x43, 0x2C, 0x32, 0x42, 0x31, 0x46, 0x42, 0x41, 0x43, 0x41, 0x38, 0x36, 0x32, 0x36, 0x33, 0x34, 0x41, 0x37, 0x0A, 0x0A, 0x61, 0x52, 0x78, 0x49, 0x58, 0x33, 0x59, 0x4D, 0x68, 0x49, 0x50, 0x41, 0x73, 0x59, 0x79, 0x6F, 0x6A, 0x49, 0x76, 0x46, 0x7A, 0x42, 0x75,

				0x6B, 0x74, 0x6B, 0x4A, 0x47, 0x5A, 0x38, 0x4D, 0x64, 0x33, 0x5A, 0x53, 0x73, 0x39, 0x41, 0x2B, 0x52, 0x2B, 0x57, 0x45, 0x59, 0x41, 0x70, 0x34, 0x63, 0x4F, 0x55, 0x43, 0x4A, 0x78, 0x51, 0x2F, 0x66, 0x4A, 0x38, 0x58, 0x4F, 0x45, 0x64, 0x58, 0x38, 0x0A, 0x31, 0x63, 0x4E, 0x66, 0x4B, 0x2B, 0x49, 0x62, 0x76, 0x4B, 0x4D, 0x68, 0x55, 0x67, 0x30, 0x4B, 0x4E, 0x35, 0x38, 0x37, 0x71, 0x66, 0x2F, 0x4C, 0x31, 0x76, 0x57, 0x58, 0x6F, 0x31, 0x74, 0x5A, 0x6B, 0x59, 0x2B, 0x5A, 0x53, 0x4E, 0x63, 0x46, 0x45, 0x41, 0x76, 0x37, 0x43, 0x43, 0x50, 0x51, 0x6B, 0x64, 0x4A, 0x42, 0x48, 0x35, 0x65, 0x6B, 0x35, 0x44, 0x51, 0x2F, 0x37, 0x6D, 0x71, 0x55, 0x0A, 0x6B, 0x76, 0x78, 0x48, 0x53, 0x50, 0x70, 0x34, 0x66, 0x41, 0x71, 0x47, 0x61, 0x68, 0x54, 0x31, 0x75, 0x37, 0x37, 0x56, 0x66, 0x4E, 0x66, 0x31, 0x53, 0x74, 0x61, 0x73, 0x31, 0x6E, 0x4F, 0x67, 0x6A, 0x50, 0x31, 0x41, 0x6C, 0x7A, 0x6E, 0x6B, 0x6A, 0x57, 0x61, 0x72,

				0x6A, 0x51, 0x4F, 0x73, 0x48, 0x46, 0x33, 0x41, 0x46, 0x31, 0x62, 0x61, 0x51, 0x4A, 0x50, 0x5A, 0x31, 0x6A, 0x71, 0x4C, 0x0A, 0x61, 0x30, 0x49, 0x45, 0x6E, 0x30, 0x6C, 0x59, 0x6C, 0x78, 0x35, 0x79, 0x4D, 0x6D, 0x78, 0x54, 0x47, 0x57, 0x79, 0x52, 0x35, 0x70, 0x57, 0x51, 0x35, 0x71, 0x66, 0x78, 0x2B, 0x62, 0x37, 0x64, 0x37, 0x75, 0x71, 0x67, 0x47, 0x69, 0x66, 0x36, 0x6A, 0x44, 0x47, 0x4D, 0x37, 0x68, 0x38, 0x43, 0x78, 0x2F, 0x74, 0x67, 0x2B, 0x61, 0x62, 0x45, 0x31, 0x34, 0x30, 0x2F, 0x50, 0x66, 0x6C, 0x33, 0x0A, 0x33, 0x6A, 0x50, 0x6C, 0x52, 0x75, 0x73, 0x57, 0x6F, 0x6F, 0x63, 0x49, 0x41, 0x76, 0x49, 0x74, 0x79, 0x51, 0x6D, 0x39, 0x39, 0x71, 0x74, 0x34, 0x64, 0x6E, 0x74, 0x6E, 0x74, 0x6F, 0x4A, 0x43, 0x6D, 0x4F, 0x53, 0x79, 0x71, 0x67, 0x4D, 0x6E, 0x76, 0x2F, 0x76, 0x2B, 0x51, 0x48, 0x74, 0x79, 0x4D, 0x73, 0x42, 0x64, 0x38, 0x34, 0x78, 0x45, 0x57, 0x46, 0x36, 0x72, 0x58, 0x4D, 0x52, 0x63, 0x0A, 0x53,

				0x2B, 0x66, 0x68, 0x54, 0x71, 0x58, 0x74, 0x54, 0x38, 0x44, 0x50, 0x65, 0x70, 0x2F, 0x56, 0x44, 0x66, 0x65, 0x78, 0x6B, 0x41, 0x63, 0x6D, 0x63, 0x75, 0x41, 0x69, 0x6F, 0x2B, 0x79, 0x64, 0x51, 0x75, 0x49, 0x31, 0x32, 0x7A, 0x50, 0x70, 0x45, 0x68, 0x50, 0x45, 0x68, 0x31, 0x44, 0x50, 0x58, 0x73, 0x64, 0x58, 0x67, 0x64, 0x77, 0x39, 0x75, 0x46, 0x47, 0x6D, 0x63, 0x35, 0x68, 0x52, 0x0A, 0x35, 0x31, 0x57, 0x41, 0x31, 0x65, 0x63, 0x44, 0x48, 0x6A, 0x31, 0x58, 0x32, 0x45, 0x72, 0x36, 0x39, 0x59, 0x70, 0x31, 0x50, 0x69, 0x43, 0x37, 0x49, 0x47, 0x79, 0x6F, 0x71, 0x57, 0x43, 0x37, 0x69, 0x2F, 0x71, 0x6D, 0x6D, 0x72, 0x49, 0x66, 0x6F, 0x41, 0x54, 0x74, 0x39, 0x58, 0x34, 0x30, 0x54, 0x56, 0x63, 0x37, 0x42, 0x63, 0x6A, 0x34, 0x63, 0x54, 0x31, 0x78, 0x37, 0x6B, 0x70, 0x4F, 0x0A, 0x4C, 0x71, 0x67, 0x33, 0x6C, 0x50, 0x78, 0x33, 0x2B, 0x4A, 0x63, 0x33, 0x43, 0x67, 0x34, 0x79, 0x5A, 0x54, 0x66, 0x6E, 0x4A, 0x5A, 0x37, 0x48, 0x76, 0x36, 0x64, 0x68,

				0x67, 0x45, 0x6D, 0x70, 0x4D, 0x73, 0x74, 0x46, 0x65, 0x35, 0x34, 0x49, 0x53, 0x76, 0x74, 0x38, 0x37, 0x59, 0x4E, 0x77, 0x74, 0x4C, 0x65, 0x6C, 0x34, 0x67, 0x50, 0x4A, 0x79, 0x53, 0x42, 0x30, 0x4B, 0x76, 0x37, 0x69, 0x0A, 0x33, 0x32, 0x74, 0x37, 0x67, 0x4F, 0x30, 0x79, 0x6D, 0x73, 0x62, 0x71, 0x4A, 0x55, 0x75, 0x79, 0x41, 0x68, 0x47, 0x64, 0x33, 0x63, 0x2B, 0x78, 0x4C, 0x46, 0x2F, 0x63, 0x63, 0x4F, 0x57, 0x44, 0x52, 0x34, 0x79, 0x72, 0x30, 0x6A, 0x79, 0x64, 0x74, 0x70, 0x79, 0x69, 0x64, 0x52, 0x45, 0x66, 0x56, 0x46, 0x66, 0x53, 0x6C, 0x39, 0x54, 0x30, 0x6D, 0x53, 0x72, 0x4E, 0x76, 0x43, 0x71, 0x45, 0x0A, 0x52, 0x52, 0x5A, 0x6E, 0x42, 0x56, 0x76, 0x37, 0x50, 0x66, 0x6C, 0x75, 0x72, 0x31, 0x59, 0x35, 0x70, 0x2F, 0x65, 0x78, 0x54, 0x63, 0x56, 0x34, 0x72, 0x4B, 0x52, 0x69, 0x6C, 0x35, 0x58, 0x6A, 0x2F, 0x39, 0x59, 0x56, 0x31, 0x4E, 0x6E, 0x6D, 0x4E, 0x2B, 0x2F, 0x31, 0x31, 0x74, 0x36, 0x58, 0x74, 0x6A, 0x72, 0x75, 0x52, 0x62,

				0x33, 0x79, 0x70, 0x38, 0x76, 0x64, 0x6C, 0x61, 0x65, 0x5A, 0x0A, 0x6C, 0x67, 0x45, 0x69, 0x73, 0x30, 0x42, 0x7A, 0x4B, 0x59, 0x39, 0x59, 0x64, 0x58, 0x48, 0x64, 0x46, 0x58, 0x57, 0x59, 0x4F, 0x41, 0x71, 0x50, 0x48, 0x45, 0x65, 0x4B, 0x57, 0x79, 0x61, 0x59, 0x5A, 0x56, 0x79, 0x43, 0x70, 0x51, 0x65, 0x43, 0x53, 0x71, 0x4F, 0x71, 0x48, 0x38, 0x67, 0x42, 0x6B, 0x4F, 0x62, 0x43, 0x69, 0x72, 0x41, 0x6A, 0x65, 0x56, 0x70, 0x35, 0x7A, 0x37, 0x6B, 0x31, 0x0A, 0x64, 0x4F, 0x2F, 0x6D, 0x56, 0x74, 0x49, 0x2B, 0x57, 0x47, 0x30, 0x48, 0x72, 0x37, 0x5A, 0x4C, 0x53, 0x52, 0x78, 0x6F, 0x61, 0x44, 0x47, 0x42, 0x33, 0x4E, 0x35, 0x38, 0x4B, 0x56, 0x45, 0x4F, 0x34, 0x65, 0x46, 0x56, 0x75, 0x6E, 0x59, 0x77, 0x51, 0x42, 0x54, 0x7A, 0x4F, 0x65, 0x57, 0x39, 0x6C, 0x4B, 0x79, 0x49, 0x38, 0x67, 0x4D, 0x45, 0x57, 0x6C, 0x62, 0x4B, 0x72, 0x41, 0x45, 0x49, 0x0A, 0x46, 0x4B, 0x38, 0x7A, 0x58, 0x6F, 0x44, 0x74, 0x39, 0x6A,

				0x7A, 0x54, 0x37, 0x67, 0x68, 0x6A, 0x79, 0x45, 0x54, 0x67, 0x44, 0x6C, 0x69, 0x50, 0x53, 0x49, 0x46, 0x6A, 0x79, 0x31, 0x64, 0x6B, 0x6A, 0x6D, 0x68, 0x53, 0x78, 0x79, 0x6A, 0x67, 0x62, 0x71, 0x45, 0x3D, 0x0A, 0x2D, 0x2D, 0x2D, 0x2D, 0x2D, 0x45, 0x4E, 0x44, 0x20, 0x52, 0x53, 0x41, 0x20, 0x50, 0x52, 0x49, 0x56, 0x41, 0x54, 0x45, 0x20, 0x4B, 0x45, 0x59, 0x2D, 0x2D, 0x2D, 0x2D, 0x2D, 0x2D, 0x0A}

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo PCD &raquo PcdsPatchableInModule &raquo common

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertifica

Private Key's size.
Private Key's size.

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdIpsecUefiCertificateKeySize	gEfiNetworkPkgTokenSpaceGuid	0x00000006	UINT32	0x3d5

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

gEfiNetworkPkgTokenSpaceGuid.PcdIpsecUefiCertifica

X509 certificate as Public Key's size.
Pubic Key's size.

PCD Information

Name	Token Space	Token number	Data Type	Default Value
PcdIpsecUefiCertificateSize	gEfiNetworkPkgTokenSpaceGuid	0x00000004	UINT32	0x251

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document

GUID

INDEX

- common

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo GUID

common

INDEX

- gEfiNetworkPkgTokenSpaceGuid
- gIp6ConfigNvDataGuid
- gIScsiConfigGuid

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

gEfiNetworkPkgTokenSpaceGuid

Network package token space guid.
Include/Guid/NetworkPkgTokenSpace.h

GUID Information

GUID's Guid Name	GUID's Guid
gEfiNetworkPkgTokenSpaceGuid	{ 0x40e064b2, 0x0ae0, 0x48b1, { 0xa0, 0x7d, 0xf8, 0xcf, 0x1e, 0x1a, 0x23, 0x10}}

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo GUID &raquo common

gIp6ConfigNvDataGuid

GUIDs used as HII FormSet and HII Package list GUID in Ip6Config driver.

Copyright (c) 2011, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Refer to Header File

NetworkPkg/Include/Guid/Ip6ConfigHii.h

GUID Information

GUID's Guid Name	GUID's Guid
gIp6ConfigNvDataGuid	{ 0x2eea107, 0x98db, 0x400e, { 0x98, 0x30, 0x46, 0xa, 0x15, 0x42, 0xd7, 0x99}}

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

gIScsiConfigGuid

GUIDs used as HII FormSet and HII Package list GUID in IScsiConfig driver that supports IP4 and IP6 both.

Copyright (c) 2011, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

GUID Information

GUID's Guid Name	GUID's Guid
gIScsiConfigGuid	{ 0x4b47d616, 0xa8d6, 0x4552, { 0x9d, 0x44, 0xcc, 0xad, 0x2e, 0xf, 0x4c, 0xf9}}

Refer to Header File

NetworkPkg/Include/Guid/IScsiConfigHii.h

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document

Modules

INDEX

- Dhcp6Dxe
- DnsDxe
- HttpBootDxe
- HttpDxe
- HttpUtilitiesDxe
- IfConfig6
- Ip6Dxe
- IpSecConfig
- IpSecDxe
- IScsiDxe
- Mtftp6Dxe
- Ping6
- TcpDxe
- Udp6Dxe
- UefiPxeBcDxe
- VConfig

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

Dhcp6Dxe

Client-side DHCPv6 services.

This driver produces EFI DHCPv6 Protocol which is used to get IPv6 addresses
and other configuration parameters from DHCPv6 servers.

(C) Copyright 2015 Hewlett-Packard Development Company, L.P.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010005
Module Guid	95E3669D-34BE-4775-A651-7EA41B69D89E
Module Type	UEFI_DRIVER
Module Version	1.0
Module Entry Point Function	Dhcp6DriverEntryPoint

Pcds

PCD Name	TokenSpace	Package
PcdDhcp6UidType	gEfiNetworkPkgTokenSpaceGuid	NetworkPkg
TianoCore	[UserExtensions	None
uni	Dhcp6DxeExtra	None

Protocols

Name	Package
gEfiUdp6ServiceBindingProtocolGuid	MdePkg
gEfiUdp6ProtocolGuid	MdePkg
gEfiDhcp6ServiceBindingProtocolGuid	MdePkg
gEfiDhcp6ProtocolGuid	MdePkg
gEfiIp6ConfigProtocolGuid	MdePkg

Library Classes

Name	Type	Package	Header File
UefiLib	Consume	MdePkg	MdePkg/Include/Library/UefiLib.h
BaseLib	Consume	MdePkg	MdePkg/Include/Library/BaseLib.h
BaseMemoryLib	Consume	MdePkg	MdePkg/Include/Library/BaseMemoryLib.h
MemoryAllocationLib	Consume	MdePkg	MdePkg/Include/Library/MemoryAllocationLib.h
UefiDriverEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiDriverEntryPoint.h
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
UefiRuntimeServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiRuntimeServicesTableLib.h
DebugLib	Consume	MdePkg	MdePkg/Include/Library/DebugLib.h
NetLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/NetLib.h
UdpIoLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/UdpIoLib.h

Source Files

- NetworkPkg/Dhcp6Dxe/Dhcp6Driver.c
- NetworkPkg/Dhcp6Dxe/Dhcp6Driver.h
- NetworkPkg/Dhcp6Dxe/Dhcp6Impl.c
- NetworkPkg/Dhcp6Dxe/Dhcp6Impl.h
- NetworkPkg/Dhcp6Dxe/Dhcp6Io.c
- NetworkPkg/Dhcp6Dxe/Dhcp6Io.h
- NetworkPkg/Dhcp6Dxe/Dhcp6Utility.c
- NetworkPkg/Dhcp6Dxe/Dhcp6Utility.h
- NetworkPkg/Dhcp6Dxe/ComponentName.c

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo Modules

DnsDxe

Implementation of EFI_DNS4_PROTOCOL and EFI_DNS6_PROTOCOL interfaces.

Copyright (c) 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010005
Module Guid	b219e140-dffc-11e3-b956-0022681e6906
Module Type	UEFI_DRIVER
Module Version	1.0
Module Entry Point Function	DnsDriverEntryPoint

Protocols

Name	Package
gEfiDns4ServiceBindingProtocolGuid	MdePkg
gEfiDns4ProtocolGuid	MdePkg
gEfiUdp4ServiceBindingProtocolGuid	MdePkg
gEfiUdp4ProtocolGuid	MdePkg
gEfiDhcp4ServiceBindingProtocolGuid	MdePkg
gEfiDhcp4ProtocolGuid	MdePkg
gEfiIp4Config2ProtocolGuid	MdePkg
gEfiIp4ServiceBindingProtocolGuid	MdePkg
gEfiIp4ProtocolGuid	MdePkg
gEfiManagedNetworkServiceBindingProtocolGuid	MdePkg
gEfiManagedNetworkProtocolGuid	MdePkg
gEfiDns6ServiceBindingProtocolGuid	MdePkg
gEfiDns6ProtocolGuid	MdePkg
gEfiUdp6ServiceBindingProtocolGuid	MdePkg

gEfiUdp6ProtocolGuid	MdePkg
gEfiDhcp6ServiceBindingProtocolGuid	MdePkg
gEfiDhcp6ProtocolGuid	MdePkg

Library Classes

Name	Type	Package	Header File
BaseLib	Consume	MdePkg	MdePkg/Include/Library/BaseLib.h
UefiLib	Consume	MdePkg	MdePkg/Include/Library/UefiLib.h
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
UefiDriverEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiDriverEntryPoint.h
UefiRuntimeServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiRuntimeServicesTableLib.h
BaseMemoryLib	Consume	MdePkg	MdePkg/Include/Library/BaseMemoryLib.h
MemoryAllocationLib	Consume	MdePkg	MdePkg/Include/Library/MemoryAllocationLib.h
NetLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/NetLib.h
DebugLib	Consume	MdePkg	MdePkg/Include/Library/DebugLib.h
DpcLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/DpcLib.h
PrintLib	Consume	MdePkg	MdePkg/Include/Library/PrintLib.h
UdpIoLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/UdpIoLib.h

Source Files

- NetworkPkg/DnsDxe/ComponentName.c
- NetworkPkg/DnsDxe/DnsDriver.h
- NetworkPkg/DnsDxe/DnsDriver.c
- NetworkPkg/DnsDxe/DnsImpl.h
- NetworkPkg/DnsDxe/DnsImpl.c
- NetworkPkg/DnsDxe/DnsProtocol.c
- NetworkPkg/DnsDxe/DnsDhcp.h
- NetworkPkg/DnsDxe/DnsDhcp.c

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo Modules

HttpBootDxe

This modules produce the Load File Protocol for UEFI HTTP boot.

Copyright (c) 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010005
Module Guid	ecebcb00-d9c8-11e4-af3d-8cdcd426c973
Module Type	UEFI_DRIVER
Module Version	1.0
Module Entry Point Function	HttpBootDxeDriverEntryPoint

Protocols

Name	Package
gEfiDevicePathProtocolGuid	MdePkg
gEfiLoadFileProtocolGuid	MdePkg
gEfiHttpServiceBindingProtocolGuid	MdePkg
gEfiHttpProtocolGuid	MdePkg
gEfiDhcp4ServiceBindingProtocolGuid	MdePkg
gEfiDhcp4ProtocolGuid	MdePkg
gEfiIp4Config2ProtocolGuid	MdePkg
gEfiNetworkInterfaceIdentifierProtocolGuid_31	MdePkg
[UserExtensions.TianoCore."ExtraFiles"]	None
HttpBootDxeExtra.uni	None

Library Classes

Name	Type	Package	Header File
UefiDriverEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiDriverEntryPoint.h
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
MemoryAllocationLib	Consume	MdePkg	MdePkg/Include/Library/MemoryAllocationLib.h

BaseLib	Consume	MdePkg	MdePkg/Include/Library/BaseLib.h
UefiLib	Consume	MdePkg	MdePkg/Include/Library/UefiLib.h
DevicePathLib	Consume	MdePkg	MdePkg/Include/Library/DevicePathLib.h
DebugLib	Consume	MdePkg	MdePkg/Include/Library/DebugLib.h
NetLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/NetLib.h
HttpLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/HttpLib.h

Source Files

- NetworkPkg/HttpBootDxe/HttpBootDxe.h
- NetworkPkg/HttpBootDxe/HttpBootDxe.c
- NetworkPkg/HttpBootDxe/HttpBootComponentName.h
- NetworkPkg/HttpBootDxe/HttpBootComponentName.c
- NetworkPkg/HttpBootDxe/HttpBootImpl.h
- NetworkPkg/HttpBootDxe/HttpBootImpl.c
- NetworkPkg/HttpBootDxe/HttpBootDhcp4.h
- NetworkPkg/HttpBootDxe/HttpBootDhcp4.c
- NetworkPkg/HttpBootDxe/HttpBootSupport.h
- NetworkPkg/HttpBootDxe/HttpBootSupport.c
- NetworkPkg/HttpBootDxe/HttpBootClient.h
- NetworkPkg/HttpBootDxe/HttpBootClient.c

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo Modules

HttpDxe

Implementation of EFI HTTP protocol interfaces.

Copyright (c) 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php.
THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010005
Module Guid	2366c20f-e15a-11e3-8bf1-e4115b28bc50
Module Type	UEFI_DRIVER
Module Version	1.0
Module Entry Point Function	HttpDxeDriverEntryPoint

Protocols

Name	Package
gEfiHttpServiceBindingProtocolGuid	MdePkg
gEfiHttpProtocolGuid	MdePkg
gEfiHttpUtilitiesProtocolGuid	MdePkg
gEfiTcp4ServiceBindingProtocolGuid	MdePkg
gEfiTcp4ProtocolGuid	MdePkg
gEfiDns4ServiceBindingProtocolGuid	MdePkg
gEfiDns4ProtocolGuid	MdePkg
gEfiIp4Config2ProtocolGuid	MdePkg
[UserExtensions.TianoCore."ExtraFiles"]	None
HttpDxeExtra.uni	None

Library Classes

Name	Type	Package	Header File
UefiDriverEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiDriverEntryPoint.h
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
MemoryAllocationLib	Consume	MdePkg	MdePkg/Include/Library/MemoryAllocationLib.h

BaseLib	Consume	MdePkg	MdePkg/Include/Library/BaseLib.h
UefiLib	Consume	MdePkg	MdePkg/Include/Library/UefiLib.h
DebugLib	Consume	MdePkg	MdePkg/Include/Library/DebugLib.h
NetLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/NetLib.h
HttpLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/HttpLib.h
DpcLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/DpcLib.h

Source Files

- NetworkPkg/HttpDxe/ComponentName.h
- NetworkPkg/HttpDxe/ComponentName.c
- NetworkPkg/HttpDxe/HttpDns.h
- NetworkPkg/HttpDxe/HttpDns.c
- NetworkPkg/HttpDxe/HttpDriver.h
- NetworkPkg/HttpDxe/HttpDriver.c
- NetworkPkg/HttpDxe/HttpImpl.h
- NetworkPkg/HttpDxe/HttpImpl.c
- NetworkPkg/HttpDxe/HttpProto.h
- NetworkPkg/HttpDxe/HttpProto.c

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo Modules

HttpUtilitiesDxe

Implementation of EFI Http Utilities Protocol interfaces.

Copyright (c) 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php.
THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010005
Module Guid	22ea234f-e72a-11e4-91f9-28d2447c4829
Module Type	DXE_DRIVER
Module Version	1.0
Module Entry Point Function	HttpUtilitiesDxeDriverEntryPoint

Protocols

Name	Package
gEfiHttpUtilitiesProtocolGuid	MdePkg

Library Classes

Name	Type	Package	Header File
UefiDriverEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiDriverEntryPoint.h
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
MemoryAllocationLib	Consume	MdePkg	MdePkg/Include/Library/MemoryAllocationLib.h
BaseMemoryLib	Consume	MdePkg	MdePkg/Include/Library/BaseMemoryLib.h
BaseLib	Consume	MdePkg	MdePkg/Include/Library/BaseLib.h
UefiLib	Consume	MdePkg	MdePkg/Include/Library/UefiLib.h
DebugLib	Consume	MdePkg	MdePkg/Include/Library/DebugLib.h

Source Files

- NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesDxe.h
- NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesDxe.c
- NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesImpl.c
- NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesProtocol.c

Module Dependencies

TRUE

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo Modules

IfConfig6

Shell application IfConfig6.

It is shell application which is used to set and get configurations for the
EFI IPv6 network stack.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010006
Module Guid	6F71926E-60CE-428d-AA58-A3D9FB879429
Module Type	UEFI_APPLICATION
Module Version	1.0
Module Entry Point Function	IfConfig6Initialize

Protocols

Name	Package
gEfiIp6ServiceBindingProtocolGuid	MdePkg
gEfiIp6ConfigProtocolGuid	MdePkg
[UserExtensions.TianoCore."ExtraFiles"]	None
IfConfig6Extra.uni	None

Library Classes

Name	Type	Package	Header File
BaseLib	Consume	MdePkg	MdePkg/Include/Library/BaseLib.h
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
UefiApplicationEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiApplicationEntryPoint.h
BaseMemoryLib	Consume	MdePkg	MdePkg/Include/Library/BaseMemoryLib.h
ShellLib	Consume	ShellPkg	ShellPkg/Include/Library/ShellLib.h
MemoryAllocationLib	Consume	MdePkg	MdePkg/Include/Library/MemoryAllocationLib.h
DebugLib	Consume	MdePkg	MdePkg/Include/Library/DebugLib.h

HiiLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/HiiLib.h
NetLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/NetLib.h

Source Files

- NetworkPkg/Application/IfConfig6/IfConfig6Strings.uni
- NetworkPkg/Application/IfConfig6/IfConfig6.c
- NetworkPkg/Application/IfConfig6/IfConfig6.h

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo Modules

Ip6Dxe

Basic IPv6 packet I/O Service.

This module provides basic network IPv6 packet I/O services which includes support for Neighbor Discovery Protocol (ND), Multicast Listener Discovery Protocol (MLD), and a subset of the Internet Control Message Protocol (ICMPv6). This driver also provides the mechanism to set and get various types of configurations for the EFI IPv6 network stack.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010005
Module Guid	5BEDB5CC-D830-4eb2-8742-2D4CC9B54F2C
Module Type	UEFI_DRIVER
Module Version	1.0
Module Entry Point Function	Ip6DriverEntryPoint

Protocols

Name	Package
gEfiManagedNetworkServiceBindingProtocolGuid	MdePkg
gEfiManagedNetworkProtocolGuid	MdePkg
gEfiIp6ServiceBindingProtocolGuid	MdePkg
gEfiIp6ProtocolGuid	MdePkg
gEfiIp6ConfigProtocolGuid	MdePkg
gEfiDhcp6ServiceBindingProtocolGuid	MdePkg
gEfiDhcp6ProtocolGuid	MdePkg
gEfiIpSec2ProtocolGuid	MdePkg
gEfiHiiConfigAccessProtocolGuid	MdePkg

Guids

Name	Package
gEfiIfrTianoGuid	MdeModulePkg
gIp6ConfigNvDataGuid	NetworkPkg
[UserExtensions.TianoCore."ExtraFiles"]	None
Ip6DxeExtra.uni	None

Library Classes

Name	Type	Package	Header File
BaseLib	Consume	MdePkg	MdePkg/Include/Library/BaseLib.h
BaseMemoryLib	Consume	MdePkg	MdePkg/Include/Library/BaseMemoryLib.h
DevicePathLib	Consume	MdePkg	MdePkg/Include/Library/DevicePathLib.h
HiiLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/HiiLib.h
UefiHiiServicesLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/UefiHiiServicesLib.h
PrintLib	Consume	MdePkg	MdePkg/Include/Library/PrintLib.h
MemoryAllocationLib	Consume	MdePkg	MdePkg/Include/Library/MemoryAllocationLib.h
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
UefiDriverEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiDriverEntryPoint.h
UefiRuntimeServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiRuntimeServicesTableLib.h
UefiLib	Consume	MdePkg	MdePkg/Include/Library/UefiLib.h
DebugLib	Consume	MdePkg	MdePkg/Include/Library/DebugLib.h
NetLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/NetLib.h
DpcLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/DpcLib.h

Source Files

- NetworkPkg/Ip6Dxe/Ip6Output.h
- NetworkPkg/Ip6Dxe/Ip6Option.h
- NetworkPkg/Ip6Dxe/Ip6Input.h
- NetworkPkg/Ip6Dxe/Ip6Nd.h
- NetworkPkg/Ip6Dxe/Ip6Mld.h
- NetworkPkg/Ip6Dxe/Ip6Impl.c
- NetworkPkg/Ip6Dxe/Ip6Driver.c
- NetworkPkg/Ip6Dxe/ComponentName.c
- NetworkPkg/Ip6Dxe/Ip6Nd.c
- NetworkPkg/Ip6Dxe/Ip6Input.c
- NetworkPkg/Ip6Dxe/Ip6ConfigImpl.c
- NetworkPkg/Ip6Dxe/Ip6ConfigImpl.h
- NetworkPkg/Ip6Dxe/Ip6Impl.h
- NetworkPkg/Ip6Dxe/Ip6Option.c
- NetworkPkg/Ip6Dxe/Ip6If.h

- NetworkPkg/Ip6Dxe/Ip6Icmp.h
- NetworkPkg/Ip6Dxe/Ip6Mld.c
- NetworkPkg/Ip6Dxe/Ip6Common.c
- NetworkPkg/Ip6Dxe/Ip6Route.c
- NetworkPkg/Ip6Dxe/Ip6If.c
- NetworkPkg/Ip6Dxe/Ip6Driver.h
- NetworkPkg/Ip6Dxe/Ip6Output.c
- NetworkPkg/Ip6Dxe/Ip6Icmp.c
- NetworkPkg/Ip6Dxe/Ip6Common.h
- NetworkPkg/Ip6Dxe/Ip6Route.h
- NetworkPkg/Ip6Dxe/Ip6DxeStrings.uni
- NetworkPkg/Ip6Dxe/Ip6NvData.h
- NetworkPkg/Ip6Dxe/Ip6ConfigNv.c
- NetworkPkg/Ip6Dxe/Ip6ConfigNv.h
- NetworkPkg/Ip6Dxe/Ip6Config.vfr

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo Modules

IpSecConfig

Shell application IpSecConfig.

This application is used to set and retrieve security and policy related information
for the EFI IPsec protocol driver.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010006
Module Guid	0922E604-F5EC-42ef-980D-A35E9A2B1844
Module Type	UEFI_APPLICATION
Module Version	1.0
Module Entry Point Function	InitializeIpSecConfig

Protocols

Name	Package
gEfiIpSec2ProtocolGuid	MdePkg
gEfiIpSecConfigProtocolGuid	MdePkg
[UserExtensions.TianoCore."ExtraFiles"]	None
IpSecConfigExtra.uni	None

Library Classes

Name	Type	Package	Header File
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
UefiApplicationEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiApplicationEntryPoint.h
BaseMemoryLib	Consume	MdePkg	MdePkg/Include/Library/BaseMemoryLib.h
ShellLib	Consume	ShellPkg	ShellPkg/Include/Library/ShellLib.h
MemoryAllocationLib	Consume	MdePkg	MdePkg/Include/Library/MemoryAllocationLib.h
DebugLib	Consume	MdePkg	MdePkg/Include/Library/DebugLib.h
HiiLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/HiiLib.h

NetLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/NetLib.h
UefiLib	Consume	MdePkg	MdePkg/Include/Library/UefiLib.h

Source Files

- NetworkPkg/Application/IpsecConfig/IpSecConfigStrings.uni
- NetworkPkg/Application/IpsecConfig/IpSecConfig.c
- NetworkPkg/Application/IpsecConfig/IpSecConfig.h
- NetworkPkg/Application/IpsecConfig/Dump.c
- NetworkPkg/Application/IpsecConfig/Dump.h
- NetworkPkg/Application/IpsecConfig/Indexer.c
- NetworkPkg/Application/IpsecConfig/Indexer.h
- NetworkPkg/Application/IpsecConfig/Match.c
- NetworkPkg/Application/IpsecConfig/Match.h
- NetworkPkg/Application/IpsecConfig/Delete.h
- NetworkPkg/Application/IpsecConfig/Delete.c
- NetworkPkg/Application/IpsecConfig/Helper.c
- NetworkPkg/Application/IpsecConfig/Helper.h
- NetworkPkg/Application/IpsecConfig/ForEach.c
- NetworkPkg/Application/IpsecConfig/ForEach.h
- NetworkPkg/Application/IpsecConfig/PolicyEntryOperation.c
- NetworkPkg/Application/IpsecConfig/PolicyEntryOperation.h

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

IpSecDxe

Packet-level security for IP datagram.

This driver provides EFI IPsec2 Protocol which is used to abstract the ability to deal with the individual packets sent and received by the host and provide packet-level security for IP datagram. It provides the IP packet protection via ESP and it supports IKEv2 for key negotiation.

Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010005
Module Guid	EE8367C0-A1D6-4565-8F89-EF628547B722
Module Type	UEFI_DRIVER
Module Version	1.0
Module Entry Point Function	IpSecDriverEntryPoint

Pcds

PCD Name	TokenSpace	Package
PcdIpsecCertificateEnabled	gEfiNetworkPkgTokenSpaceGuid	NetworkPkg
PcdIpsecUefiCaFile	gEfiNetworkPkgTokenSpaceGuid	NetworkPkg
PcdIpsecUefiCaFileSize	gEfiNetworkPkgTokenSpaceGuid	NetworkPkg
PcdIpsecUefiCertificate	gEfiNetworkPkgTokenSpaceGuid	NetworkPkg
PcdIpsecUefiCertificateSize	gEfiNetworkPkgTokenSpaceGuid	NetworkPkg
PcdIpsecUefiCertificateKey	gEfiNetworkPkgTokenSpaceGuid	NetworkPkg
PcdIpsecUefiCertificateKeySize	gEfiNetworkPkgTokenSpaceGuid	NetworkPkg
TianoCore	[UserExtensions	None
uni	IpSecDxeExtra	None

Protocols

Name	Package

gEfiIp4Config2ProtocolGuid	MdePkg
gEfiUdp4ServiceBindingProtocolGuid	MdePkg
gEfiUdp4ProtocolGuid	MdePkg
gEfiUdp6ServiceBindingProtocolGuid	MdePkg
gEfiUdp6ProtocolGuid	MdePkg
gEfiIpSecConfigProtocolGuid	MdePkg
gEfiIpSec2ProtocolGuid	MdePkg

Library Classes

Name	Type	Package	Header File
MemoryAllocationLib	Consume	MdePkg	MdePkg/Include/Library/MemoryAllocationLib.h
BaseLib	Consume	MdePkg	MdePkg/Include/Library/BaseLib.h
UefiLib	Consume	MdePkg	MdePkg/Include/Library/UefiLib.h
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
UefiRuntimeServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiRuntimeServicesTableLib.h
UefiDriverEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiDriverEntryPoint.h
BaseMemoryLib	Consume	MdePkg	MdePkg/Include/Library/BaseMemoryLib.h
DebugLib	Consume	MdePkg	MdePkg/Include/Library/DebugLib.h
PrintLib	Consume	MdePkg	MdePkg/Include/Library/PrintLib.h
BaseCryptLib	Consume	CryptoPkg	CryptoPkg/Include/Library/BaseCryptLib.h
DpcLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/DpcLib.h
UdpIoLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/UdpIoLib.h
NetLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/NetLib.h
PcdLib	Consume	MdePkg	MdePkg/Include/Library/PcdLib.h

Source Files

- NetworkPkg/IpSecDxe/IpSecConfigImpl.c
- NetworkPkg/IpSecDxe/IpSecConfigImpl.h
- NetworkPkg/IpSecDxe/IpSecCryptIo.h
- NetworkPkg/IpSecDxe/IpSecCryptIo.c
- NetworkPkg/IpSecDxe/IpSecDebug.h
- NetworkPkg/IpSecDxe/ComponentName.c
- NetworkPkg/IpSecDxe/IkeCommon.h
- NetworkPkg/IpSecDxe/IpSecImpl.c
- NetworkPkg/IpSecDxe/IkeService.c
- NetworkPkg/IpSecDxe/Ike.h
- NetworkPkg/IpSecDxe/IkePacket.h
- NetworkPkg/IpSecDxe/IkePacket.c
- NetworkPkg/IpSecDxe/IpSecDebug.c
- NetworkPkg/IpSecDxe/IpSecMain.c
- NetworkPkg/IpSecDxe/IpSecDriver.c

- NetworkPkg/IpSecDxe/IkeCommon.c
- NetworkPkg/IpSecDxe/IetfConstants.c
- NetworkPkg/IpSecDxe/IpSecImpl.h
- NetworkPkg/IpSecDxe/IkeService.h
- NetworkPkg/IpSecDxe/Ikev2/Ikev2.h
- NetworkPkg/IpSecDxe/Ikev2/Payload.h
- NetworkPkg/IpSecDxe/Ikev2/Utility.h
- NetworkPkg/IpSecDxe/Ikev2/Utility.c
- NetworkPkg/IpSecDxe/Ikev2/Sa.c
- NetworkPkg/IpSecDxe/Ikev2/ChildSa.c
- NetworkPkg/IpSecDxe/Ikev2/Info.c
- NetworkPkg/IpSecDxe/Ikev2/Payload.c
- NetworkPkg/IpSecDxe/Ikev2/Exchange.c

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo Modules

IScsiDxe

Client-side iSCSI service.

The iSCSI driver provides iSCSI service in the preboot environment and supports booting over iSCSI.

Copyright (c) 2004 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010005
Module Guid	86CDDF93-4872-4597-8AF9-A35AE4D3725F
Module Type	UEFI_DRIVER
Module Version	1.0
Module Entry Point Function	IScsiDriverEntryPoint

Protocols

Name	Package
gEfiAcpiTableProtocolGuid	MdePkg
gEfiDriverBindingProtocolGuid	MdePkg
gEfiPciIoProtocolGuid	MdePkg
gEfiDhcp4ProtocolGuid	MdePkg
gEfiDhcp6ProtocolGuid	MdePkg
gEfiDhcp4ServiceBindingProtocolGuid	MdePkg
gEfiDhcp6ServiceBindingProtocolGuid	MdePkg
gEfiTcp4ProtocolGuid	MdePkg
gEfiTcp6ProtocolGuid	MdePkg
gEfiTcp4ServiceBindingProtocolGuid	MdePkg
gEfiTcp6ServiceBindingProtocolGuid	MdePkg
gEfiExtScsiPassThruProtocolGuid	MdePkg
gEfiHiiConfigAccessProtocolGuid	MdePkg
gEfiDevicePathProtocolGuid	MdePkg

gEfiIScsiInitiatorNameProtocolGuid	MdePkg
gEfiAuthenticationInfoProtocolGuid	MdePkg

Guids

Name	Package
gEfiEventExitBootServicesGuid	MdePkg
gEfiIfrTianoGuid	MdeModulePkg
gEfiAcpiTableGuid	MdePkg
gEfiAcpi10TableGuid	MdePkg
gEfiAcpi20TableGuid	MdePkg
gIScsiConfigGuid	NetworkPkg
[UserExtensions.TianoCore."ExtraFiles"]	None
IScsiDxeExtra.uni	None

Library Classes

Name	Type	Package	Header File
BaseLib	Consume	MdePkg	MdePkg/Include/Library/BaseLib.h
BaseMemoryLib	Consume	MdePkg	MdePkg/Include/Library/BaseMemoryLib.h
DebugLib	Consume	MdePkg	MdePkg/Include/Library/DebugLib.h
DevicePathLib	Consume	MdePkg	MdePkg/Include/Library/DevicePathLib.h
HiiLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/HiiLib.h
MemoryAllocationLib	Consume	MdePkg	MdePkg/Include/Library/MemoryAllocationLib.h
NetLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/NetLib.h
TcpIoLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/TcpIoLib.h
PrintLib	Consume	MdePkg	MdePkg/Include/Library/PrintLib.h
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
UefiDriverEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiDriverEntryPoint.h
UefiLib	Consume	MdePkg	MdePkg/Include/Library/UefiLib.h
UefiRuntimeServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiRuntimeServicesTableLib.h
UefiHiiServicesLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/UefiHiiServicesLib.h
BaseCryptLib	Consume	CryptoPkg	CryptoPkg/Include/Library/BaseCryptLib.h

Source Files

- NetworkPkg/IScsiDxe/ComponentName.c
- NetworkPkg/IScsiDxe/IScsiAuthenticationInfo.c
- NetworkPkg/IScsiDxe/IScsiCHAP.h
- NetworkPkg/IScsiDxe/IScsiCHAP.c

- NetworkPkg/IScsiDxe/IScsiConfig.c
- NetworkPkg/IScsiDxe/IScsiConfig.h
- NetworkPkg/IScsiDxe/IScsiConfigNVDataStruc.h
- NetworkPkg/IScsiDxe/IScsiConfigStrings.uni
- NetworkPkg/IScsiDxe/IScsiConfigVfr.vfr
- NetworkPkg/IScsiDxe/IScsiDhcp.c
- NetworkPkg/IScsiDxe/IScsiDhcp.h
- NetworkPkg/IScsiDxe/IScsiDhcp6.c
- NetworkPkg/IScsiDxe/IScsiDhcp6.h
- NetworkPkg/IScsiDxe/IScsiDriver.c
- NetworkPkg/IScsiDxe/IScsiDriver.h
- NetworkPkg/IScsiDxe/IScsiExtScsiPassThru.c
- NetworkPkg/IScsiDxe/IScsiIbft.c
- NetworkPkg/IScsiDxe/IScsiIbft.h
- NetworkPkg/IScsiDxe/IScsiInitiatorName.c
- NetworkPkg/IScsiDxe/IScsiImpl.h
- NetworkPkg/IScsiDxe/IScsiMisc.c
- NetworkPkg/IScsiDxe/IScsiMisc.h
- NetworkPkg/IScsiDxe/IScsiProto.c
- NetworkPkg/IScsiDxe/IScsiProto.h

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

Mtftp6Dxe

Client-side Mtftp6 service.

This module produces EFI MTFTPv6 Protocol which provides basic services for client-side unicast and/or multicast TFTP.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010005
Module Guid	99F03B99-98D8-49dd-A8D3-3219D0FFE41E
Module Type	UEFI_DRIVER
Module Version	1.0
Module Entry Point Function	Mtftp6DriverEntryPoint

Protocols

Name	Package
gEfiUdp6ServiceBindingProtocolGuid	MdePkg
gEfiUdp6ProtocolGuid	MdePkg
gEfiMtftp6ServiceBindingProtocolGuid	MdePkg
gEfiMtftp6ProtocolGuid	MdePkg
[UserExtensions.TianoCore."ExtraFiles"]	None
Mtftp6DxeExtra.uni	None

Library Classes

Name	Type	Package	Header File
UefiLib	Consume	MdePkg	MdePkg/Include/Library/UefiLib.h
BaseLib	Consume	MdePkg	MdePkg/Include/Library/BaseLib.h
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
UefiRuntimeServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiRuntimeServicesTableLib
UefiDriverEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiDriverEntryPoint.h

DebugLib	Consume	MdePkg	MdePkg/Include/Library/DebugLib.h
NetLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/NetLib.h
UdpIoLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/UdpIoLib.h

Source Files

- NetworkPkg/Mtftp6Dxe/Mtftp6Driver.c
- NetworkPkg/Mtftp6Dxe/Mtftp6Driver.h
- NetworkPkg/Mtftp6Dxe/Mtftp6Impl.c
- NetworkPkg/Mtftp6Dxe/Mtftp6Impl.h
- NetworkPkg/Mtftp6Dxe/Mtftp6Option.c
- NetworkPkg/Mtftp6Dxe/Mtftp6Option.h
- NetworkPkg/Mtftp6Dxe/Mtftp6Support.h
- NetworkPkg/Mtftp6Dxe/Mtftp6Support.c
- NetworkPkg/Mtftp6Dxe/Mtftp6Rrq.c
- NetworkPkg/Mtftp6Dxe/Mtftp6Wrq.c
- NetworkPkg/Mtftp6Dxe/ComponentName.c

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo Modules

Ping6

Shell application Ping6.

It is an shell application which is used to Ping the target host with IPv6 stack.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010006
Module Guid	F35F733F-5235-4d7b-83FA-97780CEBCB20
Module Type	UEFI_APPLICATION
Module Version	1.0
Module Entry Point Function	InitializePing6

Protocols

Name	Package
gEfiCpuArchProtocolGuid	MdePkg
gEfiIp6ProtocolGuid	MdePkg
gEfiIp6ServiceBindingProtocolGuid	MdePkg
gEfiIp6ConfigProtocolGuid	MdePkg
[UserExtensions.TianoCore."ExtraFiles"]	None
Ping6Extra.uni	None

Library Classes

Name	Type	Package	Header File
BaseLib	Consume	MdePkg	MdePkg/Include/Library/BaseLib.h
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
UefiApplicationEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiApplicationEntryPoint.h
BaseMemoryLib	Consume	MdePkg	MdePkg/Include/Library/BaseMemoryLib.h
ShellLib	Consume	ShellPkg	ShellPkg/Include/Library/ShellLib.h

MemoryAllocationLib	Consume	MdePkg	MdePkg/Include/Library/MemoryAllocationLib.h
DebugLib	Consume	MdePkg	MdePkg/Include/Library/DebugLib.h
HiiLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/HiiLib.h
NetLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/NetLib.h

Source Files

- NetworkPkg/Application/Ping6/Ipf/Itc.c
- NetworkPkg/Application/Ping6/Ping6.c
- NetworkPkg/Application/Ping6/Ping6Strings.uni
- NetworkPkg/Application/Ping6/Ping6.h
- NetworkPkg/Application/Ping6/X64/Tsc.c
- NetworkPkg/Application/Ping6/Ia32/Tsc.c

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo Modules

TcpDxe

TCPv4 I/O and TCPv6 I/O services.

This module provides EFI TCPv4 Protocol and EFI TCPv6 Protocol to send and receive data stream.
It might provide TCPv4 Protocol or TCPv6 Protocol or both of them that depends on
which network stack has been loaded in system.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010005
Module Guid	1A7E4468-2F55-4a56-903C-01265EB7622B
Module Type	UEFI_DRIVER
Module Version	1.0
Module Entry Point Function	TcpDriverEntryPoint

Protocols

Name	Package
gEfiDevicePathProtocolGuid	MdePkg
gEfiIp4ProtocolGuid	MdePkg
gEfiIp4ServiceBindingProtocolGuid	MdePkg
gEfiTcp4ProtocolGuid	MdePkg
gEfiTcp4ServiceBindingProtocolGuid	MdePkg
gEfiIp6ProtocolGuid	MdePkg
gEfiIp6ServiceBindingProtocolGuid	MdePkg
gEfiTcp6ProtocolGuid	MdePkg
gEfiTcp6ServiceBindingProtocolGuid	MdePkg
[UserExtensions.TianoCore."ExtraFiles"]	None
TcpDxeExtra.uni	None

Library Classes

Name	Type	Package	Header File
BaseLib	Consume	MdePkg	MdePkg/Include/Library/BaseLib.h
BaseMemoryLib	Consume	MdePkg	MdePkg/Include/Library/BaseMemoryLib.h
DevicePathLib	Consume	MdePkg	MdePkg/Include/Library/DevicePathLib.h
DebugLib	Consume	MdePkg	MdePkg/Include/Library/DebugLib.h
MemoryAllocationLib	Consume	MdePkg	MdePkg/Include/Library/MemoryAllocationLib.h
UefiLib	Consume	MdePkg	MdePkg/Include/Library/UefiLib.h
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
UefiDriverEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiDriverEntryPoint.h
UefiRuntimeServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiRuntimeServicesTableLib.h
DpcLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/DpcLib.h
NetLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/NetLib.h
IpIoLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/IpIoLib.h

Source Files

- NetworkPkg/TcpDxe/TcpDriver.c
- NetworkPkg/TcpDxe/SockImpl.c
- NetworkPkg/TcpDxe/SockInterface.c
- NetworkPkg/TcpDxe/TcpDispatcher.c
- NetworkPkg/TcpDxe/TcpOutput.c
- NetworkPkg/TcpDxe/TcpMain.c
- NetworkPkg/TcpDxe/SockImpl.h
- NetworkPkg/TcpDxe/TcpMisc.c
- NetworkPkg/TcpDxe/TcpProto.h
- NetworkPkg/TcpDxe/TcpOption.c
- NetworkPkg/TcpDxe/TcpInput.c
- NetworkPkg/TcpDxe/TcpFunc.h
- NetworkPkg/TcpDxe/TcpOption.h
- NetworkPkg/TcpDxe/TcpTimer.c
- NetworkPkg/TcpDxe/TcpMain.h
- NetworkPkg/TcpDxe/Socket.h
- NetworkPkg/TcpDxe/ComponentName.c
- NetworkPkg/TcpDxe/TcpIo.c
- NetworkPkg/TcpDxe/TcpDriver.h

Udp6Dxe

UDP packet service based on IPv6 stack.

This module produces EFI UDPv6 Protocol which provides simple packet-oriented services to transmit and receive UDP packets.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010005
Module Guid	D912C7BC-F098-4367-92BA-E911083C7B0E
Module Type	UEFI_DRIVER
Module Version	1.0
Module Entry Point Function	Udp6DriverEntryPoint

Protocols

Name	Package
gEfiIp6ProtocolGuid	MdePkg
gEfiIp6ServiceBindingProtocolGuid	MdePkg
gEfiUdp6ServiceBindingProtocolGuid	MdePkg
gEfiUdp6ProtocolGuid	MdePkg
[UserExtensions.TianoCore."ExtraFiles"]	None
Udp6DxeExtra.uni	None

Library Classes

Name	Type	Package	Header File
BaseLib	Consume	MdePkg	MdePkg/Include/Library/BaseLib.h
BaseMemoryLib	Consume	MdePkg	MdePkg/Include/Library/BaseMemoryLib.h
MemoryAllocationLib	Consume	MdePkg	MdePkg/Include/Library/MemoryAllocationLib.h
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
UefiDriverEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiDriverEntryPoint.h

UefiRuntimeServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiRuntimeServicesTableLib.h
UefiLib	Consume	MdePkg	MdePkg/Include/Library/UefiLib.h
DebugLib	Consume	MdePkg	MdePkg/Include/Library/DebugLib.h
IpIoLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/IpIoLib.h
NetLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/NetLib.h
DpcLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/DpcLib.h

Source Files

- NetworkPkg/Udp6Dxe/Udp6Driver.h
- NetworkPkg/Udp6Dxe/Udp6Driver.c
- NetworkPkg/Udp6Dxe/Udp6Impl.c
- NetworkPkg/Udp6Dxe/Udp6Impl.h
- NetworkPkg/Udp6Dxe/ComponentName.c
- NetworkPkg/Udp6Dxe/Udp6Main.c

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo Modules

UefiPxeBcDxe

Access PXE-compatible devices for network access and network booting.

This driver provides PXE Base Code Protocol which is used to accessing PXE-compatible device for network access or booting. It could work together with an IPv4 stack, an IPv6 stack or both.

Copyright (c) 2007 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010005
Module Guid	B95E9FDA-26DE-48d2-8807-1F9107AC5E3A
Module Type	UEFI_DRIVER
Module Version	1.0
Module Entry Point Function	PxeBcDriverEntryPoint

Pcds

PCD Name	TokenSpace	Package
PcdTftpBlockSize	gEfiMdeModulePkgTokenSpaceGuid	MdeModulePkg
TianoCore	[UserExtensions	None
uni	UefiPxeBcDxeExtra	None

Protocols

Name	Package
gEfiDevicePathProtocolGuid	MdePkg
gEfiNetworkInterfaceIdentifierProtocolGuid_31	MdePkg
gEfiArpServiceBindingProtocolGuid	MdePkg
gEfiArpProtocolGuid	MdePkg
gEfiIp4ServiceBindingProtocolGuid	MdePkg
gEfiIp4ProtocolGuid	MdePkg

gEfiIp4Config2ProtocolGuid	MdePkg
gEfiIp6ServiceBindingProtocolGuid	MdePkg
gEfiIp6ProtocolGuid	MdePkg
gEfiIp6ConfigProtocolGuid	MdePkg
gEfiUdp4ServiceBindingProtocolGuid	MdePkg
gEfiUdp4ProtocolGuid	MdePkg
gEfiMtftp4ServiceBindingProtocolGuid	MdePkg
gEfiMtftp4ProtocolGuid	MdePkg
gEfiDhcp4ServiceBindingProtocolGuid	MdePkg
gEfiDhcp4ProtocolGuid	MdePkg
gEfiUdp6ServiceBindingProtocolGuid	MdePkg
gEfiUdp6ProtocolGuid	MdePkg
gEfiMtftp6ServiceBindingProtocolGuid	MdePkg
gEfiMtftp6ProtocolGuid	MdePkg
gEfiDhcp6ServiceBindingProtocolGuid	MdePkg
gEfiDhcp6ProtocolGuid	MdePkg
gEfiPxeBaseCodeCallbackProtocolGuid	MdePkg
gEfiPxeBaseCodeProtocolGuid	MdePkg
gEfiLoadFileProtocolGuid	MdePkg
gEfiAdapterInformationProtocolGuid	MdePkg

Guids

Name	Package
gEfiAdapterInfoUndiIpv6SupportGuid	MdePkg

Library Classes

Name	Type	Package	Header File
BaseLib	Consume	MdePkg	MdePkg/Include/Library/BaseLib.h
UefiLib	Consume	MdePkg	MdePkg/Include/Library/UefiLib.h
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
UefiDriverEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiDriverEntryPoint.h
BaseMemoryLib	Consume	MdePkg	MdePkg/Include/Library/BaseMemoryLib.h
MemoryAllocationLib	Consume	MdePkg	MdePkg/Include/Library/MemoryAllocationLib.h
DebugLib	Consume	MdePkg	MdePkg/Include/Library/DebugLib.h
NetLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/NetLib.h

DpcLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/DpcLib.h
DevicePathLib	Consume	MdePkg	MdePkg/Include/Library/DevicePathLib.h
PcdLib	Consume	MdePkg	MdePkg/Include/Library/PcdLib.h

Source Files

- NetworkPkg/UefiPxeBcDxe/ComponentName.c
- NetworkPkg/UefiPxeBcDxe/PxeBcDriver.c
- NetworkPkg/UefiPxeBcDxe/PxeBcDriver.h
- NetworkPkg/UefiPxeBcDxe/PxeBcImpl.c
- NetworkPkg/UefiPxeBcDxe/PxeBcImpl.h
- NetworkPkg/UefiPxeBcDxe/PxeBcBoot.c
- NetworkPkg/UefiPxeBcDxe/PxeBcBoot.h
- NetworkPkg/UefiPxeBcDxe/PxeBcDhcp6.c
- NetworkPkg/UefiPxeBcDxe/PxeBcDhcp6.h
- NetworkPkg/UefiPxeBcDxe/PxeBcDhcp4.c
- NetworkPkg/UefiPxeBcDxe/PxeBcDhcp4.h
- NetworkPkg/UefiPxeBcDxe/PxeBcMtftp.c
- NetworkPkg/UefiPxeBcDxe/PxeBcMtftp.h
- NetworkPkg/UefiPxeBcDxe/PxeBcSupport.c
- NetworkPkg/UefiPxeBcDxe/PxeBcSupport.h

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

NetworkPkg Package Document &raquo Modules

VConfig

Shell application VLAN configuration.

It is shell application which is used to get and set VLAN configuration.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials
are licensed and made available under the terms and conditions of the BSD License
which accompanies this distribution. The full text of the license may be found at
http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Basic Module Information

Version of INF file specification	0x00010005
Module Guid	87E36301-0406-44db-AAF3-9E0E591F3725
Module Type	UEFI_APPLICATION
Module Version	1.0
Module Entry Point Function	VlanConfigMain

Protocols

Name	Package
gEfiVlanConfigProtocolGuid	MdePkg
[UserExtensions.TianoCore."ExtraFiles"]	None
VConfigExtra.uni	None

Library Classes

Name	Type	Package	Header File
UefiApplicationEntryPoint	Consume	MdePkg	MdePkg/Include/Library/UefiApplicationEntryPoint.h
UefiBootServicesTableLib	Consume	MdePkg	MdePkg/Include/Library/UefiBootServicesTableLib.h
UefiLib	Consume	MdePkg	MdePkg/Include/Library/UefiLib.h
ShellLib	Consume	ShellPkg	ShellPkg/Include/Library/ShellLib.h
NetLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/NetLib.h
MemoryAllocationLib	Consume	MdePkg	MdePkg/Include/Library/MemoryAllocationLib.h
HiiLib	Consume	MdeModulePkg	MdeModulePkg/Include/Library/HiiLib.h

Source Files

- NetworkPkg/Application/VConfig/VConfigStrings.uni
- NetworkPkg/Application/VConfig/VConfig.c

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

Data Structures

Here are the data structures with brief descriptions:

_ARG_LIST
_AUTH_ALGORITHM
_DHCP6_IA_CB
_DHCP6_INF_CB
_DHCP6_INSTANCE
_DHCP6_SERVICE
_DHCP6_TX_CB
_DNS_DRIVER_DATA
_DNS_FLAGS
_DNS_INSTANCE
_DNS_SERVICE
_EFI_AH_HEADER
_EFI_ESP_HEADER
_EFI_ESP_TAIL
_ENCRYPT_ALGORITHM
_HASH_ALGORITHM
_HTTP_BOOT_PRIVATE_DATA
_HTTP_PROTOCOL
_HTTP_SERVICE
_ICMP6_ECHO_REQUEST_REPLY
_IFCONFIG6_INTERFACE_CB
_IFCONFIG6_PRIVATE_DATA
_IKE_EXCHANGE_INTERFACE
_IP6_ADDRESS_INFO
_IP6_CONFIG_INSTANCE
_IP6_DAD_ENTRY
_IP6_DEFAULT_ROUTER
_IP6_DELAY_JOIN_LIST
_IP6_ETHE_ADDR_OPTION
_IP6_FORM_CALLBACK_INFO
_IP6_FRAGMENT_HEADER
_IP6_INTERFACE
_IP6_MTU_OPTION
_IP6_NEIGHBOR_ENTRY
_IP6_PREFIX_INFO_OPTION
_IP6_PREFIX_LIST_ENTRY
_IP6_PROTOCOL
_IP6_ROUTE_TABLE
_IP6_ROUTING_HEADER
_IP6_SERVICE
_IPSEC_PAD_ENTRY

_IPSEC_PRIVATE_DATA
_IPSEC_RECYCLE_CONTEXT
_IPSEC_SAD_DATA
_IPSEC_SAD_ENTRY
_IPSEC_SPD_DATA
_IPSEC_SPD_ENTRY
_ISCSI_ADDTIONAL_HEADER
_ISCSI_ASYNC_MESSAGE
_ISCSI_ATTEMPT_CONFIG_NVDATA
_ISCSI_BASIC_HEADER
_ISCSI_BI_EXP_READ_DATA_LEN_AHS
_ISCSI_CHAP_AUTH_CONFIG_NVDATA
_ISCSI_CHAP_AUTH_DATA
_ISCSI_CONFIG_IFR_NVDATA
_ISCSI_CONNECTION
_ISCSI_DRIVER_DATA
_ISCSI_FORM_CALLBACK_INFO
_ISCSI_IN_BUFFER_CONTEXT
_ISCSI_KEY_VALUE_PAIR
_ISCSI_LOGIN_REQUEST
_ISCSI_LOGIN_RESPONSE
_ISCSI_LOGOUT_REQUEST
_ISCSI_LOGOUT_RESPONSE
_ISCSI_NOP_IN
_ISCSI_NOP_OUT
_ISCSI_PRIVATE_PROTOCOL
_ISCSI_READY_TO_TRANSFER
_ISCSI_REJECT
_ISCSI_ROOT_PATH_FIELD
_ISCSI_SCSI_DATA_IN
_ISCSI_SCSI_DATA_OUT
_ISCSI_SENSE_DATA
_ISCSI_SESSION
_ISCSI_SESSION_CONFIG_NVDATA
_ISCSI_SNACK_REQUEST
_ISCSI_TCB
_ISCSI_TMF_REQUEST
_ISCSI_TMF_RESPONSE
_ISCSI_XFER_CONTEXT
_MTFTP6_GETINFO_CONTEXT
_MTFTP6_INSTANCE
_MTFTP6_SERVICE
_NET_PROTOCOL
_PING6_ICMP6_TX_INFO
_PING6_PRIVATE_DATA

_PXEBC_PRIVATE_DATA
_PXEBC_PRIVATE_PROTOCOL
_PXEBC_VIRTUAL_NIC
_SCSI_COMMAND
_SCSI_RESPONSE
_SOCK_BUFFER
_SOCK_COMPLETION_TOKEN
_SOCK_INIT_DATA
_SOCK_IO_TOKEN
_SOCK_TOKEN
_TCP4_MODE_DATA
_TCP4_ROUTE_INFO
_TCP6_MODE_DATA
_TCP_CONTROL_BLOCK
_TCP_HEARTBEAT_TIMER
_TCP_OPTION
_TCP_PEER
_TCP_PROTO_DATA
_TCP_RSV_DATA
_TCP_SEG
_TCP_SERVICE_DATA
_TCP_SOCKET
_UDP6_INSTANCE_DATA
_UDP6_RXDATA_WRAP
_UDP6_SERVICE_DATA
_VAR_CHECK_ITEM
DELETE_POLICY_ENTRY_CONTEXT
DNS4_CACHE
DNS4_SERVER_INFOR
DNS4_SERVER_IP
DNS4_TOKEN_ENTRY
DNS6_CACHE
DNS6_SERVER_INFOR
DNS6_SERVER_IP
DNS6_TOKEN_ENTRY
DNS_ANSWER_SECTION
DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT
DNS_HEADER
DNS_QUERY_SECTION
EDIT_POLICY_ENTRY_CONTEXT
HASH_DATA_FRAGMENT
HII_VENDOR_DEVICE_PATH
HTTP4_IO_CONFIG_DATA
HTTP_BOOT_CACHE_CONTENT
HTTP_BOOT_CALLBACK_DATA

HTTP_BOOT_DHCP4_OPTION_ARCH
HTTP_BOOT_DHCP4_OPTION_CLID
HTTP_BOOT_DHCP4_OPTION_ENTRY
HTTP_BOOT_DHCP4_OPTION_MAX_MESG_SIZE
HTTP_BOOT_DHCP4_OPTION_MESG
HTTP_BOOT_DHCP4_OPTION_PARA
HTTP_BOOT_DHCP4_OPTION_UNDI
HTTP_BOOT_DHCP4_OPTION_UUID
HTTP_BOOT_DHCP4_PACKET
HTTP_BOOT_DHCP4_PACKET_CACHE
HTTP_BOOT_DHCP_PACKET_CACHE
HTTP_BOOT_ENTITY_DATA
HTTP_BOOT_OPTION_BOOT_ITEM
HTTP_BOOT_URI_CONTENT
HTTP_BOOT_VENDOR_OPTION
HTTP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT
HTTP_IO
HTTP_IO_CONFIG_DATA
HTTP_IO_HEADER
HTTP_IO_RESOPNSE_DATA
HTTP_TCP_TOKEN_WRAP
HTTP_TOKEN_WRAP
IKE_ALG_GUID_INFO
IKE_HEADER
IKE_PACKET
IKE_PAYLOAD
IKE_SA_ATTR_UNION
IKE_SA_ATTRIBUTE
IKE_UDP_SERVICE
IKEV2_AUTH
IKEV2_CERT
IKEV2_CERT_REQ
IKEV2_CFG
IKEV2_CFG_ATTRIBUTES
IKEV2_CHILD_SA_SESSION
IKEV2_COMMON_PAYLOAD_HEADER
IKEV2_DELETE
IKEV2_DH_BUFFER
IKEV2_ENCRYPTED
IKEV2_ID
IKEV2_INFO_EXCHANGE_CONTEXT
IKEV2_KEY_EXCHANGE
IKEV2_NONCE
IKEV2_NOTIFY
IKEV2_PACKET_HANDLER

IKEV2_PAD_LEN
IKEV2_PROPOSAL
IKEV2_PROPOSAL_DATA
IKEV2_SA
IKEV2_SA_DATA
IKEV2_SA_PARAMS
IKEV2_SA_SESSION
IKEV2_SESSION_COMMON
IKEV2_SESSION_KEYS
IKEV2_TRANSFORM
IKEV2_TRANSFORM_DATA
IKEV2_TS
IKEV2_VENDOR
INSERT_POLICY_ENTRY_CONTEXT
IP6_ADDRESS_INFO_ENTRY
IP6_ASSEMBLE_ENTRY
IP6_ASSEMBLE_TABLE
IP6_CLIP_INFO
IP6_CONFIG_DATA
IP6_CONFIG_DATA_ITEM
IP6_CONFIG_DATA_RECORD
IP6_CONFIG_IFR_NVDATA
IP6_CONFIG_NVDATA
IP6_CONFIG_VARIABLE
IP6_DESTROY_CHILD_BY_ADDR_CALLBACK_CONTEXT
IP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT
IP6_IPSEC_WRAP
IP6_LINK_RX_TOKEN
IP6_LINK_TX_TOKEN
IP6_MLD_GROUP
IP6_MLD_HEAD
IP6_MLD_SERVICE_DATA
IP6_ROUTE_CACHE
IP6_ROUTE_CACHE_ENTRY
IP6_ROUTE_ENTRY
IP6_RXDATA_WRAP
IP6_TXTOKEN_WRAP
IP_SEC_VARIABLE_INFO
IPSEC_COMMON_POLICY_ENTRY
IPSEC_VAR_ITEM_HEADER
IPSEC_VARIABLE_BUFFER
ISCSI_NIC_INFO
ISCSI_PRIVATE_DATA
MODP_GROUP
MTFTP6_BLOCK_RANGE

MTFTP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT
MTFTP6_EXT_OPTION_INFO
PAD_ENTRY_INDEXER
POLICY_ENTRY_INDEXER
PRF_DATA_FRAGMENT
PXEBC_BOOT_MENU_ENTRY
PXEBC_BOOT_SVR_ENTRY
PXEBC_CLASS_ID
PXEBC_DHCP4_OPTION_ARCH
PXEBC_DHCP4_OPTION_CLID
PXEBC_DHCP4_OPTION_ENTRY
PXEBC_DHCP4_OPTION_MAX_MESG_SIZE
PXEBC_DHCP4_OPTION_MESG
PXEBC_DHCP4_OPTION_PARA
PXEBC_DHCP4_OPTION_UNDI
PXEBC_DHCP4_OPTION_UUID
PXEBC_DHCP4_PACKET
PXEBC_DHCP4_PACKET_CACHE
PXEBC_DHCP6_OPTION_ARCH
PXEBC_DHCP6_OPTION_ENTRY
PXEBC_DHCP6_OPTION_NODE
PXEBC_DHCP6_OPTION_ORO
PXEBC_DHCP6_OPTION_UNDI
PXEBC_DHCP6_OPTION_VENDOR_CLASS
PXEBC_DHCP6_PACKET
PXEBC_DHCP6_PACKET_CACHE
PXEBC_DHCP_PACKET_CACHE
PXEBC_MENU_PROMPT
PXEBC_OPTION_BOOT_ITEM
PXEBC_VENDOR_OPTION
SA_KEYMATS
SAD_ENTRY_INDEXER
SHELL_PARAM_PACKAGE
SOCK_IO_DATA
SPD_ENTRY_INDEXER
STR2INT
TCP_ACCESS_POINT
TCP_CONFIG_DATA
TCP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT
TRAFFIC_SELECTOR
UDP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT
VAR_CHECK_ITEM

- Data Structures
- Files

- Data Structures
- Data Fields

_ARG_LIST Struct Reference

Data Fields

ARG_LIST * Next
CHAR16 * Arg

Field Documentation

CHAR16* _ARG_LIST::Arg

Referenced by IfConfig6Initialize(), IfConfig6ParseGwDnsAddressList(), IfConfig6ParseManualAddressList(), and SplitStrToList().

ARG_LIST* _ARG_LIST::Next

Referenced by IfConfig6Cleanup(), IfConfig6Initialize(), IfConfig6ParseGwDnsAddressList(), IfConfig6ParseManualAddressList(), and SplitStrToList().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_AUTH_ALGORITHM Struct Reference

Data Fields

UINT8	AlgorithmId
UINTN	DigestLength
UINTN	IcvLength
UINTN	BlockSize
CRYPTO_HMAC_GETCONTEXTSIZE	HmacGetContextSize
CRYPTO_HMAC_INIT	HmacInitiate
CRYPTO_HMAC_UPDATE	HmacUpdate
CRYPTO_HMAC_FINAL	HmacFinal

Field Documentation

UINT8 _AUTH_ALGORITHM::AlgorithmId

UINTN _AUTH_ALGORITHM::BlockSize

UINTN _AUTH_ALGORITHM::DigestLength

Referenced by IpSecGetHmacDigestLength().

CRYPTO_HMAC_FINAL _AUTH_ALGORITHM::HmacFinal

CRYPTO_HMAC_GETCONTEXTSIZE _AUTH_ALGORITHM::HmacGetContextSize

Referenced by IpSecCryptoIoHmac().

CRYPTO_HMAC_INIT _AUTH_ALGORITHM::HmacInitiate

CRYPTO_HMAC_UPDATE _AUTH_ALGORITHM::HmacUpdate

UINTN _AUTH_ALGORITHM::IcvLength

Referenced by IpSecGetIcvLength().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_DHCP6_IA_CB Struct Reference

Data Fields

EFI_DHCP6_IA * Ia
UINT32 T1
UINT32 T2
UINT32 AllExpireTime
UINT32 LeaseTime

Field Documentation

UINT32 _DHCP6_IA_CB::AllExpireTime

Referenced by Dhcp6EnqueueRetry().

EFI_DHCP6_IA* _DHCP6_IA_CB::Ia

Referenced by Dhcp6OnTimerTick(), EfiDhcp6Configure(), EfiDhcp6Decline(), EfiDhcp6GetModeData(), EfiDhcp6Release(), EfiDhcp6RenewRebind(), EfiDhcp6Start(), and EfiDhcp6Stop().

UINT32 _DHCP6_IA_CB::LeaseTime

Referenced by Dhcp6OnTimerTick().

UINT32 _DHCP6_IA_CB::T1

Referenced by Dhcp6EnqueueRetry(), and Dhcp6OnTimerTick().

UINT32 _DHCP6_IA_CB::T2

Referenced by Dhcp6EnqueueRetry(), and Dhcp6OnTimerTick().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_DHCP6_INF_CB Struct Reference

Data Fields

LIST_ENTRY	Link
UINT32	Xid
EFI_DHCP6_INFO_CALLBACK	ReplyCallback
VOID *	CallbackContext
EFI_EVENT	TimeoutEvent

Field Documentation

VOID* _DHCP6_INF_CB::CallbackContext

Referenced by Dhcp6HandleStateless(), and Dhcp6StartInfoRequest().

LIST_ENTRY _DHCP6_INF_CB::Link

Referenced by Dhcp6CleanupRetry(), Dhcp6DequeueRetry(), and Dhcp6StartInfoRequest().

EFI_DHCP6_INFO_CALLBACK _DHCP6_INF_CB::ReplyCallback

Referenced by Dhcp6HandleStateless(), and Dhcp6StartInfoRequest().

EFI_EVENT _DHCP6_INF_CB::TimeoutEvent

Referenced by Dhcp6CleanupRetry(), Dhcp6DequeueRetry(), and Dhcp6StartInfoRequest().

UINT32 _DHCP6_INF_CB::Xid

Referenced by Dhcp6DequeueRetry(), and Dhcp6HandleStateless().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_DHCP6_INSTANCE Struct Reference

Data Fields

UINT32	Signature
EFI_HANDLE	Handle
DHCP6_SERVICE *	Service
LIST_ENTRY	Link
EFI_DHCP6_PROTOCOL	Dhcp6
EFI_EVENT	Timer
EFI_DHCP6_CONFIG_DATA *	Config
EFI_DHCP6_IA *	CacheIa
DHCP6_IA_CB	IaCb
LIST_ENTRY	TxList
LIST_ENTRY	InfList
EFI_DHCP6_PACKET *	AdSelect
UINT8	AdPref
EFI_IPv6_ADDRESS *	Unicast
volatile EFI_STATUS	UdpSts
BOOLEAN	InDestroy
BOOLEAN	MediaPresent
UINT64	StartTime

Field Documentation

UINT8 _DHCP6_INSTANCE::AdPref

Referenced by Dhcp6OnTimerTick().

EFI_DHCP6_PACKET* _DHCP6_INSTANCE::AdSelect

Referenced by Dhcp6OnTimerTick().

EFI_DHCP6_IA* _DHCP6_INSTANCE::CacheIa

EFI_DHCP6_CONFIG_DATA* _DHCP6_INSTANCE::Config

Referenced by Dhcp6OnTimerTick(), EfiDhcp6Configure(), EfiDhcp6Decline(), EfiDhcp6GetModeData(), EfiDhcp6Release(), EfiDhcp6RenewRebind(), EfiDhcp6Start(), and EfiDhcp6Stop().

EFI_DHCP6_PROTOCOL _DHCP6_INSTANCE::Dhcp6

Referenced by Dhcp6CreateInstance(), and Dhcp6ServiceBindingCreateChild().

EFI_HANDLE _DHCP6_INSTANCE::Handle

Referenced by Dhcp6DestroyChildEntry(), and Dhcp6ServiceBindingCreateChild().

DHCP6_IA_CB _DHCP6_INSTANCE::IaCb

Referenced by Dhcp6OnTimerTick(), EfiDhcp6Configure(), EfiDhcp6Decline(), EfiDhcp6GetModeData(), EfiDhcp6Release(), EfiDhcp6RenewRebind(), EfiDhcp6Start(), and EfiDhcp6Stop().

BOOLEAN _DHCP6_INSTANCE::InDestroy

Referenced by Dhcp6CreateInstance(), and Dhcp6ServiceBindingDestroyChild().

LIST_ENTRY _DHCP6_INSTANCE::InfList

Referenced by Dhcp6CreateInstance().

LIST_ENTRY _DHCP6_INSTANCE::Link

Referenced by Dhcp6ServiceBindingCreateChild(), and Dhcp6ServiceBindingDestroyChild().

BOOLEAN _DHCP6_INSTANCE::MediaPresent

Referenced by Dhcp6CreateInstance().

DHCP6_SERVICE* _DHCP6_INSTANCE::Service

Referenced by Dhcp6CreateInstance(), Dhcp6ServiceBindingDestroyChild(), EfiDhcp6Configure(), EfiDhcp6Decline(), EfiDhcp6GetModeData(), EfiDhcp6InfoRequest(), EfiDhcp6Release(), EfiDhcp6RenewRebind(), EfiDhcp6Start(), and EfiDhcp6Stop().

UINT32 _DHCP6_INSTANCE::Signature

Referenced by Dhcp6CreateInstance().

UINT64 _DHCP6_INSTANCE::StartTime

EFI_EVENT _DHCP6_INSTANCE::Timer

Referenced by Dhcp6CreateInstance(), and Dhcp6ServiceBindingCreateChild().

LIST_ENTRY _DHCP6_INSTANCE::TxList

Referenced by Dhcp6CreateInstance(), Dhcp6OnTimerTick(), and Dhcp6ReceivePacket().

volatile EFI_STATUS _DHCP6_INSTANCE::UdpSts

Referenced by Dhcp6CreateInstance(), Dhcp6OnTimerTick(), EfiDhcp6Decline(), EfiDhcp6InfoRequest(), EfiDhcp6Release(), EfiDhcp6RenewRebind(), EfiDhcp6Start(), and EfiDhcp6Stop().

EFI_IPv6_ADDRESS* _DHCP6_INSTANCE::Unicast

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_DHCP6_SERVICE Struct Reference

Data Fields

UINT32	Signature
EFI_HANDLE	Controller
EFI_HANDLE	Image
EFI_SERVICE_BINDING_PROTOCOL	ServiceBinding
EFI_SIMPLE_NETWORK_PROTOCOL *	Snp
EFI_IP6_CONFIG_PROTOCOL *	Ip6Cfg
EFI_DHCP6_DUID *	ClientId
UDP_IO *	UdpIo
UINT32	Xid
LIST_ENTRY	Child
UINTN	NumOfChild

Field Documentation

LIST_ENTRY _DHCP6_SERVICE::Child

Referenced by Dhcp6CreateService(), Dhcp6DriverBindingStop(), Dhcp6ReceivePacket(), Dhcp6ServiceBindingCreateChild(), and EfiDhcp6Configure().

EFI_DHCP6_DUID* _DHCP6_SERVICE::ClientId

Referenced by Dhcp6CreateService(), Dhcp6HandleStateful(), Dhcp6SendConfirmMsg(), Dhcp6SendDeclineMsg(), Dhcp6SendInfoRequestMsg(), Dhcp6SendReleaseMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6SendRequestMsg(), Dhcp6SendSolicitMsg(), and EfiDhcp6GetModeData().

EFI_HANDLE _DHCP6_SERVICE::Controller

Referenced by Dhcp6CreateService().

EFI_HANDLE _DHCP6_SERVICE::Image

Referenced by Dhcp6CreateService().

EFI_IP6_CONFIG_PROTOCOL* _DHCP6_SERVICE::Ip6Cfg

Referenced by Dhcp6CreateService(), and EfiDhcp6InfoRequest().

UINTN _DHCP6_SERVICE::NumOfChild

Referenced by Dhcp6ServiceBindingCreateChild(), and Dhcp6ServiceBindingDestroyChild().

EFI_SERVICE_BINDING_PROTOCOL _DHCP6_SERVICE::ServiceBinding

Referenced by Dhcp6CreateService(), and Dhcp6DriverBindingStart().

UINT32 _DHCP6_SERVICE::Signature

Referenced by Dhcp6CreateService().

EFI_SIMPLE_NETWORK_PROTOCOL* _DHCP6_SERVICE::Snp

Referenced by Dhcp6CreateService().

UDP_IO* _DHCP6_SERVICE::UdpIo

Referenced by Dhcp6CreateService(), Dhcp6HandleStateful(), Dhcp6HandleStateless(), Dhcp6ReceivePacket(), Dhcp6ServiceBindingCreateChild(), Dhcp6ServiceBindingDestroyChild(), Dhcp6StartInfoRequest(), Dhcp6TransmitPacket(), EfiDhcp6Decline(), EfiDhcp6InfoRequest(), EfiDhcp6Release(), EfiDhcp6RenewRebind(), EfiDhcp6Start(), and EfiDhcp6Stop().

UINT32 _DHCP6_SERVICE::Xid

Referenced by Dhcp6CreateService(), Dhcp6SendConfirmMsg(), Dhcp6SendDeclineMsg(), Dhcp6SendInfoRequestMsg(), Dhcp6SendReleaseMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6SendRequestMsg(), and Dhcp6SendSolicitMsg().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_DHCP6_TX_CB Struct Reference

Data Fields

LIST_ENTRY	Link
UINT32	Xid
EFI_DHCP6_PACKET *	TxPacket
EFI_DHCP6_RETRANSMISSION	RetryCtl
UINT32	RetryCnt
UINT32	RetryExp
UINT32	RetryLos
UINT32	TickTime
UINT16 *	Elapsed
BOOLEAN	SolicitRetry

Field Documentation

UINT16* _DHCP6_TX_CB::Elapsed

Referenced by Dhcp6EnqueueRetry(), and Dhcp6OnTimerTick().

LIST_ENTRY _DHCP6_TX_CB::Link

Referenced by Dhcp6CleanupRetry(), Dhcp6DequeueRetry(), and Dhcp6EnqueueRetry().

UINT32 _DHCP6_TX_CB::RetryCnt

Referenced by Dhcp6OnTimerTick().

EFI_DHCP6_RETRANSMISSION _DHCP6_TX_CB::RetryCtl

Referenced by Dhcp6EnqueueRetry(), and Dhcp6OnTimerTick().

UINT32 _DHCP6_TX_CB::RetryExp

Referenced by Dhcp6EnqueueRetry(), and Dhcp6OnTimerTick().

UINT32 _DHCP6_TX_CB::RetryLos

Referenced by Dhcp6OnTimerTick().

BOOLEAN _DHCP6_TX_CB::SolicitRetry

Referenced by Dhcp6OnTimerTick().

UINT32 _DHCP6_TX_CB::TickTime

Referenced by Dhcp6OnTimerTick().

EFI_DHCP6_PACKET* _DHCP6_TX_CB::TxPacket

Referenced by Dhcp6CleanupRetry(), Dhcp6DequeueRetry(), Dhcp6EnqueueRetry(), Dhcp6OnTimerTick(), and Dhcp6ReceivePacket().

UINT32 _DHCP6_TX_CB::Xid

Referenced by Dhcp6DequeueRetry(), Dhcp6EnqueueRetry(), Dhcp6OnTimerTick(), and Dhcp6ReceivePacket().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_DNS_DRIVER_DATA Struct Reference

Data Fields

EFI_EVENT Timer
LIST_ENTRY Dns4CacheList
Ticking timer for DNS cache update.
LIST_ENTRY Dns4ServerList
LIST_ENTRY Dns6CacheList
LIST_ENTRY Dns6ServerList

Field Documentation

LIST_ENTRY _DNS_DRIVER_DATA::Dns4CacheList
Ticking timer for DNS cache update.

Referenced by Dns4GetModeData(), Dns4HostNameToIp(), Dns4UpdateDnsCache(), DnsDriverEntryPoint(), DnsOnTimerUpdate(), DnsUnload(), and ParseDnsResponse().

LIST_ENTRY _DNS_DRIVER_DATA::Dns4ServerList

Referenced by Dns4Configure(), Dns4GetModeData(), DnsDriverEntryPoint(), and DnsUnload().

LIST_ENTRY _DNS_DRIVER_DATA::Dns6CacheList

Referenced by Dns6GetModeData(), Dns6HostNameToIp(), Dns6UpdateDnsCache(), DnsDriverEntryPoint(), DnsOnTimerUpdate(), DnsUnload(), and ParseDnsResponse().

LIST_ENTRY _DNS_DRIVER_DATA::Dns6ServerList

Referenced by Dns6Configure(), Dns6GetModeData(), DnsDriverEntryPoint(), and DnsUnload().

EFI_EVENT _DNS_DRIVER_DATA::Timer

Referenced by DnsDriverEntryPoint(), and DnsUnload().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_DNS_FLAGS Union Reference

Data Fields

```
struct {
  UINT16  RCode:4
  UINT16  Zero:3
  UINT16  RA:1
  UINT16  RD:1
  UINT16  TC:1
  UINT16  AA:1
  UINT16  OpCode:4
  UINT16  QR:1
}                    Bits
        UINT16 Uint16
```

Field Documentation

UINT16 _DNS_FLAGS::AA

struct { ... } _DNS_FLAGS::Bits

Referenced by ConstructDNSQueryIp(), and ParseDnsResponse().

UINT16 _DNS_FLAGS::OpCode

Referenced by ConstructDNSQueryIp().

UINT16 _DNS_FLAGS::QR

Referenced by ConstructDNSQueryIp(), and ParseDnsResponse().

UINT16 _DNS_FLAGS::RA

UINT16 _DNS_FLAGS::RCode

Referenced by ParseDnsResponse().

UINT16 _DNS_FLAGS::RD

Referenced by ConstructDNSQueryIp().

UINT16 _DNS_FLAGS::TC

UINT16 _DNS_FLAGS::Uint16

Referenced by ConstructDNSQueryIp(), and ParseDnsResponse().

UINT16 _DNS_FLAGS::Zero

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_DNS_INSTANCE Struct Reference

Data Fields

UINT32	Signature
LIST_ENTRY	Link
EFI_DNS4_PROTOCOL	Dns4
EFI_DNS6_PROTOCOL	Dns6
INTN	State
BOOLEAN	InDestroy
DNS_SERVICE *	Service
EFI_HANDLE	ChildHandle
EFI_DNS4_CONFIG_DATA	Dns4CfgData
EFI_DNS6_CONFIG_DATA	Dns6CfgData
EFI_IP_ADDRESS	SessionDnsServer
NET_MAP	Dns4TxTokens
NET_MAP	Dns6TxTokens
UINT32	MaxRetry
UDP_IO *	UdpIo

Field Documentation

EFI_HANDLE _DNS_INSTANCE::ChildHandle

Referenced by Dns4ServiceBindingCreateChild(), Dns6ServiceBindingCreateChild(), and DnsDestroyChildEntryInHandleBuffer().

EFI_DNS4_PROTOCOL _DNS_INSTANCE::Dns4

Referenced by Dns4ServiceBindingCreateChild(), and DnsCreateInstance().

EFI_DNS4_CONFIG_DATA _DNS_INSTANCE::Dns4CfgData

Referenced by Dns4Configure(), Dns4GetModeData(), and Dns4HostNameToIp().

NET_MAP _DNS_INSTANCE::Dns4TxTokens

Referenced by Dns4Configure(), Dns4HostNameToIp(), DnsCreateInstance(), DnsOnPacketSent(), and DnsOnTimerRetransmit().

EFI_DNS6_PROTOCOL _DNS_INSTANCE::Dns6

Referenced by Dns6ServiceBindingCreateChild(), and DnsCreateInstance().

EFI_DNS6_CONFIG_DATA _DNS_INSTANCE::Dns6CfgData

Referenced by Dns6Configure(), Dns6GetModeData(), and Dns6HostNameToIp().

NET_MAP _DNS_INSTANCE::Dns6TxTokens

Referenced by Dns6Configure(), Dns6HostNameToIp(), DnsCreateInstance(), DnsOnPacketSent(), and DnsOnTimerRetransmit().

BOOLEAN _DNS_INSTANCE::InDestroy

Referenced by Dns4ServiceBindingDestroyChild(), Dns6ServiceBindingDestroyChild(), and DnsCreateInstance().

LIST_ENTRY _DNS_INSTANCE::Link

Referenced by Dns4ServiceBindingCreateChild(), Dns4ServiceBindingDestroyChild(), Dns6ServiceBindingCreateChild(), Dns6ServiceBindingDestroyChild(), and DnsCreateInstance().

UINT32 _DNS_INSTANCE::MaxRetry

Referenced by Dns4Configure(), Dns4HostNameToIp(), Dns6Configure(), Dns6HostNameToIp(), and DnsOnTimerRetransmit().

DNS_SERVICE* _DNS_INSTANCE::Service

Referenced by Dns4ServiceBindingDestroyChild(), Dns6Configure(), Dns6ServiceBindingDestroyChild(), DnsCreateInstance(), and DnsOnPacketSent().

EFI_IP_ADDRESS _DNS_INSTANCE::SessionDnsServer

Referenced by Dns4Configure(), and Dns6Configure().

UINT32 _DNS_INSTANCE::Signature

Referenced by DnsCreateInstance().

INTN _DNS_INSTANCE::State

Referenced by Dns4Cancel(), Dns4Configure(), Dns4GetModeData(), Dns4HostNameToIp(), Dns4Poll(), Dns6Cancel(), Dns6Configure(), Dns6GetModeData(), Dns6HostNameToIp(), Dns6Poll(), and DnsCreateInstance().

UDP_IO* _DNS_INSTANCE::UdpIo

Referenced by Dns4Configure(), Dns4Poll(), Dns4ServiceBindingCreateChild(), Dns4ServiceBindingDestroyChild(), Dns6Configure(), Dns6Poll(), Dns6ServiceBindingCreateChild(), Dns6ServiceBindingDestroyChild(), DnsCreateInstance(), and DnsOnPacketReceived().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_DNS_SERVICE Struct Reference

Data Fields

UINT32	Signature
EFI_SERVICE_BINDING_PROTOCOL	ServiceBinding
UINT16	Dns4ChildrenNum
LIST_ENTRY	Dns4ChildrenList
UINT16	Dns6ChildrenNum
LIST_ENTRY	Dns6ChildrenList
EFI_HANDLE	ControllerHandle
EFI_HANDLE	ImageHandle
EFI_EVENT	TimerToGetMap
EFI_EVENT	Timer
UINT8	IpVersion
	Ticking timer for packet retransmission.
UDP_IO *	ConnectUdp

Field Documentation

UDP_IO* _DNS_SERVICE::ConnectUdp

Referenced by Dns4ServiceBindingCreateChild(), Dns4ServiceBindingDestroyChild(), Dns6ServiceBindingCreateChild(), Dns6ServiceBindingDestroyChild(), and DnsCreateService().

EFI_HANDLE _DNS_SERVICE::ControllerHandle

Referenced by Dns6Configure(), and DnsCreateService().

LIST_ENTRY _DNS_SERVICE::Dns4ChildrenList

Referenced by Dns4DriverBindingStop(), Dns4ServiceBindingCreateChild(), DnsCreateService(), and DnsOnTimerRetransmit().

UINT16 _DNS_SERVICE::Dns4ChildrenNum

Referenced by Dns4ServiceBindingCreateChild(), Dns4ServiceBindingDestroyChild(), and DnsCreateService().

LIST_ENTRY _DNS_SERVICE::Dns6ChildrenList

Referenced by Dns6DriverBindingStop(), Dns6ServiceBindingCreateChild(), DnsCreateService(), and DnsOnTimerRetransmit().

UINT16 _DNS_SERVICE::Dns6ChildrenNum

Referenced by Dns6ServiceBindingCreateChild(), Dns6ServiceBindingDestroyChild(), and DnsCreateService().

EFI_HANDLE _DNS_SERVICE::ImageHandle

Referenced by Dns6Configure(), and DnsCreateService().

UINT8 _DNS_SERVICE::IpVersion
Ticking timer for packet retransmission.

Referenced by DnsCreateService(), DnsOnPacketSent(), and DnsOnTimerRetransmit().

EFI_SERVICE_BINDING_PROTOCOL _DNS_SERVICE::ServiceBinding

Referenced by Dns4DriverBindingStart(), Dns6DriverBindingStart(), and DnsCreateService().

UINT32 _DNS_SERVICE::Signature

Referenced by DnsCreateService().

EFI_EVENT _DNS_SERVICE::Timer

Referenced by Dns4DriverBindingStart(), Dns6DriverBindingStart(), and DnsCreateService().

EFI_EVENT _DNS_SERVICE::TimerToGetMap

Referenced by Dns4GetMapping(), Dns6GetMapping(), and DnsCreateService().

_EFI_AH_HEADER Struct Reference

Data Fields

UINT8 NextHeader
UINT8 PayloadLen
UINT16 Reserved
UINT32 Spi
UINT32 SequenceNumber

Field Documentation

UINT8 _EFI_AH_HEADER::NextHeader

UINT8 _EFI_AH_HEADER::PayloadLen

UINT16 _EFI_AH_HEADER::Reserved

UINT32 _EFI_AH_HEADER::SequenceNumber

UINT32 _EFI_AH_HEADER::Spi

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_EFI_ESP_HEADER Struct Reference

Data Fields

UINT32 Spi
UINT32 SequenceNumber

Field Documentation

UINT32 _EFI_ESP_HEADER::SequenceNumber

Referenced by IpSecEspOutboundPacket().

UINT32 _EFI_ESP_HEADER::Spi

Referenced by IpSecEspInboundPacket(), and IpSecEspOutboundPacket().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_EFI_ESP_TAIL Struct Reference

Data Fields

UINT8 PaddingLength
UINT8 NextHeader

Field Documentation

UINT8 _EFI_ESP_TAIL::NextHeader

Referenced by IpSecEspInboundPacket(), and IpSecEspOutboundPacket().

UINT8 _EFI_ESP_TAIL::PaddingLength

Referenced by IpSecEspInboundPacket(), and IpSecEspOutboundPacket().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_ENCRYPT_ALGORITHM Struct Reference

Data Fields

UINT8	AlgorithmId
UINTN	KeyLength
UINTN	IvLength
UINTN	BlockSize
CRYPTO_CIPHER_GETCONTEXTSIZE	CipherGetContextSize
CRYPTO_CIPHER_INIT	CipherInitiate
CRYPTO_CIPHER_ENCRYPT	CipherEncrypt
CRYPTO_CIPHER_DECRYPT	CipherDecrypt

Field Documentation

UINT8 _ENCRYPT_ALGORITHM::AlgorithmId

UINTN _ENCRYPT_ALGORITHM::BlockSize

Referenced by IpSecGetEncryptBlockSize().

CRYPTO_CIPHER_DECRYPT _ENCRYPT_ALGORITHM::CipherDecrypt

CRYPTO_CIPHER_ENCRYPT _ENCRYPT_ALGORITHM::CipherEncrypt

CRYPTO_CIPHER_GETCONTEXTSIZE _ENCRYPT_ALGORITHM::CipherGetContextSize

Referenced by IpSecCryptoIoDecrypt(), and IpSecCryptoIoEncrypt().

CRYPTO_CIPHER_INIT _ENCRYPT_ALGORITHM::CipherInitiate

UINTN _ENCRYPT_ALGORITHM::IvLength

Referenced by IpSecGetEncryptIvLength().

UINTN _ENCRYPT_ALGORITHM::KeyLength

Referenced by IpSecGetEncryptKeyLength().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page

- Data Structures
- Files

- Data Structures
- Data Fields

_HASH_ALGORITHM Struct Reference

Data Fields

UINT8	AlgorithmId
UINTN	DigestLength
UINTN	IcvLength
UINTN	BlockSize
CRYPTO_HASH_GETCONTEXTSIZE	HashGetContextSize
CRYPTO_HASH_INIT	HashInitiate
CRYPTO_HASH_UPDATE	HashUpdate
CRYPTO_HASH_FINAL	HashFinal

Field Documentation

UINT8 _HASH_ALGORITHM::AlgorithmId

UINTN _HASH_ALGORITHM::BlockSize

UINTN _HASH_ALGORITHM::DigestLength

CRYPTO_HASH_FINAL _HASH_ALGORITHM::HashFinal

CRYPTO_HASH_GETCONTEXTSIZE _HASH_ALGORITHM::HashGetContextSize

Referenced by IpSecCryptoIoHash().

CRYPTO_HASH_INIT _HASH_ALGORITHM::HashInitiate

CRYPTO_HASH_UPDATE _HASH_ALGORITHM::HashUpdate

UINTN _HASH_ALGORITHM::IcvLength

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_HTTP_BOOT_PRIVATE_DATA Struct Reference

Data Fields

UINT32	Signature
EFI_HANDLE	Controller
EFI_HANDLE	Image
EFI_HANDLE	Dhcp4Child
HTTP_IO	HttpIo
BOOLEAN	HttpCreated
EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL *	Nii
EFI_IP4_CONFIG2_PROTOCOL *	Ip4Config2
EFI_DHCP4_PROTOCOL *	Dhcp4
EFI_DEVICE_PATH_PROTOCOL *	ParentDevicePath
EFI_HANDLE	ChildHandle
EFI_LOAD_FILE_PROTOCOL	LoadFile
EFI_DEVICE_PATH_PROTOCOL *	DevicePath
UINT32	Id
BOOLEAN	UsingIpv6
BOOLEAN	Started
EFI_IP_ADDRESS	StationIp
EFI_IP_ADDRESS	SubnetMask
EFI_IP_ADDRESS	GatewayIp
UINT16	Port
CHAR8 *	BootFileUri
VOID *	BootFileUriParser
UINTN	BootFileSize
LIST_ENTRY	CacheList
UINT32	SelectIndex
UINT32	SelectProxyType
HTTP_BOOT_DHCP_PACKET_CACHE	OfferBuffer [HTTP_BOOT_OFFER_MAX_NUM]
UINT32	OfferNum
UINT32	OfferCount [HttpOfferTypeMax]
UINT32	OfferIndex [HttpOfferTypeMax][HTTP_BOOT_OFFER_MAX_

Field Documentation

UINTN _HTTP_BOOT_PRIVATE_DATA::BootFileSize

CHAR8* _HTTP_BOOT_PRIVATE_DATA::BootFileUri

VOID* _HTTP_BOOT_PRIVATE_DATA::BootFileUriParser

LIST_ENTRY _HTTP_BOOT_PRIVATE_DATA::CacheList

Referenced by HttpBootIp4DxeDriverBindingStart().

EFI_HANDLE _HTTP_BOOT_PRIVATE_DATA::ChildHandle

Referenced by HttpBootIp4DxeDriverBindingStart().

EFI_HANDLE _HTTP_BOOT_PRIVATE_DATA::Controller

Referenced by HttpBootDxeLoadFile(), HttpBootIp4DxeDriverBindingStart(), and HttpBootIp4DxeDriverBindingStop().

EFI_DEVICE_PATH_PROTOCOL* _HTTP_BOOT_PRIVATE_DATA::DevicePath

Referenced by HttpBootIp4DxeDriverBindingStart().

EFI_DHCP4_PROTOCOL* _HTTP_BOOT_PRIVATE_DATA::Dhcp4

Referenced by HttpBootDxeLoadFile(), and HttpBootIp4DxeDriverBindingStart().

EFI_HANDLE _HTTP_BOOT_PRIVATE_DATA::Dhcp4Child

Referenced by HttpBootIp4DxeDriverBindingStart().

EFI_IP_ADDRESS _HTTP_BOOT_PRIVATE_DATA::GatewayIp

BOOLEAN _HTTP_BOOT_PRIVATE_DATA::HttpCreated

HTTP_IO _HTTP_BOOT_PRIVATE_DATA::HttpIo

UINT32 _HTTP_BOOT_PRIVATE_DATA::Id

Referenced by HttpBootIp4DxeDriverBindingStart(), and HttpBootIp4DxeDriverBindingStop().

EFI_HANDLE _HTTP_BOOT_PRIVATE_DATA::Image

Referenced by HttpBootIp4DxeDriverBindingStart().

EFI_IP4_CONFIG2_PROTOCOL* _HTTP_BOOT_PRIVATE_DATA::Ip4Config2

Referenced by HttpBootIp4DxeDriverBindingStart().

EFI_LOAD_FILE_PROTOCOL _HTTP_BOOT_PRIVATE_DATA::LoadFile

Referenced by HttpBootIp4DxeDriverBindingStart().

EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL* _HTTP_BOOT_PRIVATE_DATA::Nii

Referenced by HttpBootIp4DxeDriverBindingStart().

HTTP_BOOT_DHCP_PACKET_CACHE _HTTP_BOOT_PRIVATE_DATA::OfferBuffer[HTTP_BOOT_OFFER_MAX_NUM]

Referenced by HttpBootDhcp4CallBack().

UINT32 _HTTP_BOOT_PRIVATE_DATA::OfferCount[HttpOfferTypeMax]

UINT32 _HTTP_BOOT_PRIVATE_DATA::OfferIndex[HttpOfferTypeMax][HTTP_BOOT_OFFER_MAX_NUM]

UINT32 _HTTP_BOOT_PRIVATE_DATA::OfferNum

Referenced by HttpBootDhcp4CallBack().

EFI_DEVICE_PATH_PROTOCOL* _HTTP_BOOT_PRIVATE_DATA::ParentDevicePath

Referenced by HttpBootIp4DxeDriverBindingStart().

UINT16 _HTTP_BOOT_PRIVATE_DATA::Port

UINT32 _HTTP_BOOT_PRIVATE_DATA::SelectIndex

Referenced by HttpBootDhcp4CallBack().

UINT32 _HTTP_BOOT_PRIVATE_DATA::SelectProxyType

UINT32 _HTTP_BOOT_PRIVATE_DATA::Signature

Referenced by HttpBootIp4DxeDriverBindingStart().

BOOLEAN _HTTP_BOOT_PRIVATE_DATA::Started

EFI_IP_ADDRESS _HTTP_BOOT_PRIVATE_DATA::StationIp

EFI_IP_ADDRESS _HTTP_BOOT_PRIVATE_DATA::SubnetMask

BOOLEAN _HTTP_BOOT_PRIVATE_DATA::UsingIpv6

Referenced by HttpBootIp4DxeDriverBindingStart().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_HTTP_PROTOCOL Struct Reference

Data Fields

UINT32	Signature
EFI_HTTP_PROTOCOL	Http
EFI_HANDLE	Handle
HTTP_SERVICE *	Service
LIST_ENTRY	Link
BOOLEAN	InDestroy
INTN	State
EFI_HANDLE	TcpChildHandle
EFI_TCP4_PROTOCOL *	Tcp4
EFI_TCP4_CONFIG_DATA	Tcp4CfgData
EFI_TCP4_OPTION	Tcp4Option
EFI_TCP4_CONNECTION_TOKEN	ConnToken
BOOLEAN	IsConnDone
EFI_TCP4_CLOSE_TOKEN	CloseToken
BOOLEAN	IsCloseDone
CHAR8 *	RemoteHost
UINT16	RemotePort
EFI_IPv4_ADDRESS	RemoteAddr
EFI_TCP4_IO_TOKEN	RxToken
EFI_TCP4_RECEIVE_DATA	RxData
BOOLEAN	IsRxDone
CHAR8 *	CacheBody
CHAR8 *	NextMsg
UINTN	CacheLen
UINTN	CacheOffset
VOID *	MsgParser
EFI_HTTP_VERSION	HttpVersion
UINT32	TimeOutMillisec
BOOLEAN	LocalAddressIsIPv6
EFI_HTTPv4_ACCESS_POINT	IPv4Node
NET_MAP	TxTokens
NET_MAP	RxTokens
CHAR8 *	Url

Field Documentation

CHAR8* _HTTP_PROTOCOL::CacheBody

Referenced by HttpResponseWorker(), and HttpTcpReceiveNotifyDpc().

UINTN _HTTP_PROTOCOL::CacheLen

Referenced by HttpResponseWorker(), and HttpTcpReceiveNotifyDpc().

UINTN _HTTP_PROTOCOL::CacheOffset

Referenced by HttpResponseWorker(), and HttpTcpReceiveNotifyDpc().

EFI_TCP4_CLOSE_TOKEN _HTTP_PROTOCOL::CloseToken

EFI_TCP4_CONNECTION_TOKEN _HTTP_PROTOCOL::ConnToken

EFI_HANDLE _HTTP_PROTOCOL::Handle

Referenced by HttpDestroyChildEntryInHandleBuffer(), and HttpServiceBindingCreateChild().

EFI_HTTP_PROTOCOL _HTTP_PROTOCOL::Http

Referenced by HttpServiceBindingCreateChild().

EFI_HTTP_VERSION _HTTP_PROTOCOL::HttpVersion

Referenced by EfiHttpConfigure(), and EfiHttpGetModeData().

BOOLEAN _HTTP_PROTOCOL::InDestroy

Referenced by HttpServiceBindingDestroyChild().

EFI_HTTPv4_ACCESS_POINT _HTTP_PROTOCOL::IPv4Node

Referenced by EfiHttpConfigure(), and EfiHttpGetModeData().

BOOLEAN _HTTP_PROTOCOL::IsCloseDone

BOOLEAN _HTTP_PROTOCOL::IsConnDone

BOOLEAN _HTTP_PROTOCOL::IsRxDone

Referenced by HttpResponseWorker().

LIST_ENTRY _HTTP_PROTOCOL::Link

Referenced by HttpServiceBindingCreateChild(), and HttpServiceBindingDestroyChild().

BOOLEAN _HTTP_PROTOCOL::LocalAddressIsIPv6

Referenced by EfiHttpConfigure(), EfiHttpGetModeData(), EfiHttpPoll(), EfiHttpRequest(), and EfiHttpResponse().

VOID* _HTTP_PROTOCOL::MsgParser

Referenced by HttpResponseWorker(), and HttpTcpReceiveNotifyDpc().

CHAR8* _HTTP_PROTOCOL::NextMsg

Referenced by HttpBodyParserCallback(), HttpResponseWorker(), and HttpTcpReceiveNotifyDpc().

EFI_IPv4_ADDRESS _HTTP_PROTOCOL::RemoteAddr

Referenced by EfiHttpRequest().

CHAR8* _HTTP_PROTOCOL::RemoteHost

Referenced by EfiHttpRequest().

UINT16 _HTTP_PROTOCOL::RemotePort

Referenced by EfiHttpRequest().

EFI_TCP4_RECEIVE_DATA _HTTP_PROTOCOL::RxData

EFI_TCP4_IO_TOKEN _HTTP_PROTOCOL::RxToken

Referenced by HttpResponseWorker().

NET_MAP _HTTP_PROTOCOL::RxTokens

Referenced by EfiHttpResponse(), and HttpTcpReceiveNotifyDpc().

HTTP_SERVICE* _HTTP_PROTOCOL::Service

Referenced by EfiHttpConfigure(), and HttpServiceBindingDestroyChild().

UINT32 _HTTP_PROTOCOL::Signature

INTN _HTTP_PROTOCOL::State

Referenced by EfiHttpCancel(), EfiHttpConfigure(), EfiHttpGetModeData(), EfiHttpPoll(), EfiHttpRequest(), and EfiHttpResponse().

EFI_TCP4_PROTOCOL* _HTTP_PROTOCOL::Tcp4

Referenced by EfiHttpPoll(), EfiHttpRequest(), and HttpResponseWorker().

EFI_TCP4_CONFIG_DATA _HTTP_PROTOCOL::Tcp4CfgData

EFI_TCP4_OPTION _HTTP_PROTOCOL::Tcp4Option

EFI_HANDLE _HTTP_PROTOCOL::TcpChildHandle

UINT32 _HTTP_PROTOCOL::TimeOutMillisec

Referenced by EfiHttpConfigure(), and EfiHttpGetModeData().

NET_MAP _HTTP_PROTOCOL::TxTokens

Referenced by EfiHttpRequest(), HttpResponseWorker(), and HttpTcpTransmitNotifyDpc().

CHAR8* _HTTP_PROTOCOL::Url

Referenced by EfiHttpRequest().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_HTTP_SERVICE Struct Reference

Data Fields

UINT32	Signature
EFI_SERVICE_BINDING_PROTOCOL	ServiceBinding
EFI_HANDLE	ImageHandle
EFI_HANDLE	ControllerHandle
LIST_ENTRY	ChildrenList
UINTN	ChildrenNumber
EFI_HANDLE	TcpChildHandle
INTN	State

Field Documentation

LIST_ENTRY _HTTP_SERVICE::ChildrenList

Referenced by HttpCreateService(), HttpDxeDriverBindingStop(), and HttpServiceBindingCreateChild().

UINTN _HTTP_SERVICE::ChildrenNumber

Referenced by HttpCreateService(), HttpServiceBindingCreateChild(), and HttpServiceBindingDestroyChild().

EFI_HANDLE _HTTP_SERVICE::ControllerHandle

Referenced by HttpCreateService(), and HttpDns4().

EFI_HANDLE _HTTP_SERVICE::ImageHandle

Referenced by HttpCreateService(), and HttpDns4().

EFI_SERVICE_BINDING_PROTOCOL _HTTP_SERVICE::ServiceBinding

Referenced by HttpCreateService(), and HttpDxeDriverBindingStart().

UINT32 _HTTP_SERVICE::Signature

Referenced by HttpCreateService().

INTN _HTTP_SERVICE::State

EFI_HANDLE _HTTP_SERVICE::TcpChildHandle

Referenced by HttpDxeDriverBindingStart(), HttpServiceBindingCreateChild(), and HttpServiceBindingDestroyChild().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ICMP6_ECHO_REQUEST_REPLY Struct Reference

Data Fields

UINT8 Type
UINT8 Code
UINT16 Checksum
UINT16 Identifier
UINT16 SequenceNum
UINT64 TimeStamp
UINT8 Data [1]

Field Documentation

UINT16 _ICMP6_ECHO_REQUEST_REPLY::Checksum

Referenced by Ping6GenerateToken().

UINT8 _ICMP6_ECHO_REQUEST_REPLY::Code

Referenced by Ping6GenerateToken(), and Ping6OnEchoReplyReceived().

UINT8 _ICMP6_ECHO_REQUEST_REPLY::Data[1]

UINT16 _ICMP6_ECHO_REQUEST_REPLY::Identifier

Referenced by Ping6GenerateToken().

UINT16 _ICMP6_ECHO_REQUEST_REPLY::SequenceNum

Referenced by Ping6GenerateToken(), and Ping6OnEchoReplyReceived().

UINT64 _ICMP6_ECHO_REQUEST_REPLY::TimeStamp

Referenced by Ping6GenerateToken(), and Ping6OnEchoReplyReceived().

UINT8 _ICMP6_ECHO_REQUEST_REPLY::Type

Referenced by Ping6GenerateToken(), and Ping6OnEchoReplyReceived().

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_IFCONFIG6_INTERFACE_CB Struct Reference

Data Fields

EFI_HANDLE	NicHandle
LIST_ENTRY	Link
EFI_IP6_CONFIG_PROTOCOL *	IfCfg
EFI_IP6_CONFIG_INTERFACE_INFO *	IfInfo
EFI_IP6_CONFIG_INTERFACE_ID *	IfId
EFI_IP6_CONFIG_POLICY	Policy
EFI_IP6_CONFIG_DUP_ADDR_DETECT_TRANSMITS	Xmits
UINT32	DnsCnt
EFI_IPv6_ADDRESS	DnsAddr [1]

Field Documentation

EFI_IPv6_ADDRESS _IFCONFIG6_INTERFACE_CB::DnsAddr[1]

Referenced by IfConfig6GetInterfaceInfo(), and IfConfig6ShowInterfaceInfo().

UINT32 _IFCONFIG6_INTERFACE_CB::DnsCnt

Referenced by IfConfig6GetInterfaceInfo(), and IfConfig6ShowInterfaceInfo().

EFI_IP6_CONFIG_PROTOCOL* _IFCONFIG6_INTERFACE_CB::IfCfg

Referenced by IfConfig6ClearInterfaceInfo(), IfConfig6GetInterfaceInfo(), and IfConfig6SetInterfaceInfo().

EFI_IP6_CONFIG_INTERFACE_ID* _IFCONFIG6_INTERFACE_CB::IfId

Referenced by IfConfig6Cleanup(), IfConfig6GetInterfaceInfo(), and IfConfig6ShowInterfaceInfo().

EFI_IP6_CONFIG_INTERFACE_INFO* _IFCONFIG6_INTERFACE_CB::IfInfo

Referenced by IfConfig6Cleanup(), IfConfig6GetInterfaceInfo(), and IfConfig6ShowInterfaceInfo().

LIST_ENTRY _IFCONFIG6_INTERFACE_CB::Link

Referenced by IfConfig6Cleanup(), and IfConfig6GetInterfaceInfo().

EFI_HANDLE _IFCONFIG6_INTERFACE_CB::NicHandle

Referenced by IfConfig6GetInterfaceInfo().

EFI_IP6_CONFIG_POLICY _IFCONFIG6_INTERFACE_CB::Policy

Referenced by IfConfig6GetInterfaceInfo(), and IfConfig6ShowInterfaceInfo().

EFI_IP6_CONFIG_DUP_ADDR_DETECT_TRANSMITS _IFCONFIG6_INTERFACE_CB::Xmits

Referenced by IfConfig6GetInterfaceInfo(), and IfConfig6ShowInterfaceInfo().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_IFCONFIG6_PRIVATE_DATA Struct Reference

Data Fields

EFI_HANDLE ImageHandle
LIST_ENTRY IfList
UINT32 OpCode
CHAR16 * IfName
ARG_LIST * VarArg

Field Documentation

LIST_ENTRY _IFCONFIG6_PRIVATE_DATA::IfList

Referenced by IfConfig6Initialize().

CHAR16* _IFCONFIG6_PRIVATE_DATA::IfName

Referenced by IfConfig6Initialize().

EFI_HANDLE _IFCONFIG6_PRIVATE_DATA::ImageHandle

UINT32 _IFCONFIG6_PRIVATE_DATA::OpCode

Referenced by IfConfig6Initialize().

ARG_LIST* _IFCONFIG6_PRIVATE_DATA::VarArg

Referenced by IfConfig6Initialize().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_IKE_EXCHANGE_INTERFACE Struct Reference

Data Fields

UINT8 IkeVer
IKE_NEGOTIATE_SA NegotiateSa
IKE_NEGOTIATE_CHILD_SA NegotiateChildSa
IKE_NEGOTIATE_INFO NegotiateInfo
IKE_HANDLE_SA HandleSa
IKE_HANDLE_CHILD_SA HandleChildSa
IKE_HANDLE_INFO HandleInfo

Field Documentation

IKE_HANDLE_CHILD_SA _IKE_EXCHANGE_INTERFACE::HandleChildSa

Referenced by IkeDispatch().

IKE_HANDLE_INFO _IKE_EXCHANGE_INTERFACE::HandleInfo

Referenced by IkeDispatch().

IKE_HANDLE_SA _IKE_EXCHANGE_INTERFACE::HandleSa

Referenced by IkeDispatch().

UINT8 _IKE_EXCHANGE_INTERFACE::IkeVer

IKE_NEGOTIATE_CHILD_SA _IKE_EXCHANGE_INTERFACE::NegotiateChildSa

Referenced by IkeNegotiate().

IKE_NEGOTIATE_INFO _IKE_EXCHANGE_INTERFACE::NegotiateInfo

Referenced by IkeDeleteAllSas(), and Ikev2LifetimeNotify().

IKE_NEGOTIATE_SA _IKE_EXCHANGE_INTERFACE::NegotiateSa

Referenced by IkeNegotiate().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page

- Data Structures
- Files

- Data Structures
- Data Fields

_IP6_ADDRESS_INFO Struct Reference

Data Fields

UINT32	Signature
LIST_ENTRY	Link
EFI_IPv6_ADDRESS	Address
BOOLEAN	IsAnycast
UINT8	PrefixLength
UINT32	ValidLifetime
UINT32	PreferredLifetime

Field Documentation

EFI_IPv6_ADDRESS _IP6_ADDRESS_INFO::Address

Referenced by Ip6BuildEfiAddressList(), Ip6CandidateSource(), Ip6ConfigOnDhcp6Event(), Ip6ConfigOnPolicyChanged(), Ip6ConfigSetMaunualAddress(), Ip6FindDADEntry(), Ip6IsOneOfSetAddress(), Ip6NdFasterTimerTicking(), Ip6OnDADFinished(), Ip6RemoveAddr(), Ip6SelectSourceAddress(), and Ip6SetAddress().

BOOLEAN _IP6_ADDRESS_INFO::IsAnycast

Referenced by Ip6CandidateSource(), Ip6ConfigSetMaunualAddress(), and Ip6SetAddress().

LIST_ENTRY _IP6_ADDRESS_INFO::Link

Referenced by Ip6CandidateSource(), Ip6ConfigInitInstance(), Ip6ConfigOnPolicyChanged(), and Ip6ConfigSetMaunualAddress().

UINT32 _IP6_ADDRESS_INFO::PreferredLifetime

Referenced by Ip6CandidateSource(), Ip6OnDADFinished(), and Ip6SetAddress().

UINT8 _IP6_ADDRESS_INFO::PrefixLength

Referenced by Ip6BuildEfiAddressList(), Ip6CandidateSource(), Ip6ConfigOnDhcp6Event(), Ip6ConfigProtocol(), Ip6ConfigSetMaunualAddress(), Ip6OnDADFinished(), Ip6RemoveAddr(), and Ip6SetAddress().

UINT32 _IP6_ADDRESS_INFO::Signature

Referenced by Ip6CandidateSource(), and Ip6SetAddress().

UINT32 _IP6_ADDRESS_INFO::ValidLifetime

Referenced by Ip6CandidateSource(), Ip6OnDADFinished(), and Ip6SetAddress().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_IP6_CONFIG_INSTANCE Struct Reference

Data Fields

UINT32	Signature
BOOLEAN	Configured
LIST_ENTRY	Link
UINT16	IfIndex
EFI_IP6_CONFIG_INTERFACE_INFO	InterfaceInfo
EFI_IP6_CONFIG_INTERFACE_ID	AltIfId
EFI_IP6_CONFIG_POLICY	Policy
EFI_IP6_CONFIG_DUP_ADDR_DETECT_TRANSMITS	DadXmits
IP6_CONFIG_DATA_ITEM	DataItem [Ip6ConfigDataTypeMaximum]
NET_MAP	DadFailedMap
NET_MAP	DadPassedMap
EFI_IP6_CONFIG_PROTOCOL	Ip6Config
EFI_EVENT	Dhcp6SbNotifyEvent
VOID *	Registration
EFI_HANDLE	Dhcp6Handle
EFI_DHCP6_PROTOCOL *	Dhcp6
BOOLEAN	OtherInfoOnly
UINT32	IaId
EFI_EVENT	Dhcp6Event
UINT32	FailedIaAddressCount
EFI_IPv6_ADDRESS *	DeclineAddress
UINT32	DeclineAddressCount
IP6_FORM_CALLBACK_INFO	CallbackInfo
IP6_CONFIG_NVDATA	Ip6NvData

Field Documentation

EFI_IP6_CONFIG_INTERFACE_ID _IP6_CONFIG_INSTANCE::AltIfId

IP6_FORM_CALLBACK_INFO _IP6_CONFIG_INSTANCE::CallbackInfo

BOOLEAN _IP6_CONFIG_INSTANCE::Configured

NET_MAP _IP6_CONFIG_INSTANCE::DadFailedMap

Referenced by Ip6ManualAddrDadCallback().

NET_MAP _IP6_CONFIG_INSTANCE::DadPassedMap

Referenced by Ip6ManualAddrDadCallback().

EFI_IP6_CONFIG_DUP_ADDR_DETECT_TRANSMITS _IP6_CONFIG_INSTANCE::DadXmits

Referenced by Ip6InitDADProcess().

IP6_CONFIG_DATA_ITEM _IP6_CONFIG_INSTANCE::DataItem[Ip6ConfigDataTypeMaximum]

Referenced by EfiIp6ConfigGetData(), EfiIp6ConfigRegisterDataNotify(), EfiIp6ConfigSetData(), EfiIp6ConfigUnregisterDataNotify(), Ip6CreateService(), and Ip6ManualAddrDadCallback().

EFI_IPv6_ADDRESS* _IP6_CONFIG_INSTANCE::DeclineAddress

Referenced by Ip6ConfigOnDhcp6Event(), and Ip6ConfigSetStatefulAddrCallback().

UINT32 _IP6_CONFIG_INSTANCE::DeclineAddressCount

Referenced by Ip6ConfigOnDhcp6Event(), and Ip6ConfigSetStatefulAddrCallback().

EFI_DHCP6_PROTOCOL* _IP6_CONFIG_INSTANCE::Dhcp6

Referenced by Ip6ConfigOnDhcp6Event(), Ip6ConfigSetStatefulAddrCallback(), and Ip6OnDADFinished().

EFI_EVENT _IP6_CONFIG_INSTANCE::Dhcp6Event

Referenced by Ip6DriverBindingStop(), and Ip6OnDADFinished().

EFI_HANDLE _IP6_CONFIG_INSTANCE::Dhcp6Handle

Referenced by Ip6ConfigOnDhcp6SbInstalled().

EFI_EVENT _IP6_CONFIG_INSTANCE::Dhcp6SbNotifyEvent

UINT32 _IP6_CONFIG_INSTANCE::FailedIaAddressCount

Referenced by Ip6ConfigOnDhcp6Event(), and Ip6ConfigSetStatefulAddrCallback().

UINT32 _IP6_CONFIG_INSTANCE::IaId

UINT16 _IP6_CONFIG_INSTANCE::IfIndex

Referenced by Ip6ConfigInitInstance().

EFI_IP6_CONFIG_INTERFACE_INFO _IP6_CONFIG_INSTANCE::InterfaceInfo

EFI_IP6_CONFIG_PROTOCOL _IP6_CONFIG_INSTANCE::Ip6Config

Referenced by Ip6DriverBindingStart(), and Ip6DriverBindingStop().

IP6_CONFIG_NVDATA _IP6_CONFIG_INSTANCE::Ip6NvData

Referenced by Ip6FormCallback().

LIST_ENTRY _IP6_CONFIG_INSTANCE::Link

BOOLEAN _IP6_CONFIG_INSTANCE::OtherInfoOnly

Referenced by Ip6ConfigOnDhcp6Event(), and Ip6ConfigOnDhcp6SbInstalled().

EFI_IP6_CONFIG_POLICY _IP6_CONFIG_INSTANCE::Policy

Referenced by Ip6ConfigOnDhcp6Event(), Ip6ConfigOnDhcp6SbInstalled(), and Ip6NdFasterTimerTicking().

VOID* _IP6_CONFIG_INSTANCE::Registration

UINT32 _IP6_CONFIG_INSTANCE::Signature

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_IP6_DAD_ENTRY Struct Reference

Data Fields

UINT32	Signature
LIST_ENTRY	Link
UINT32	MaxTransmit
UINT32	Transmit
UINT32	Receive
UINT32	RetransTick
IP6_ADDRESS_INFO *	AddressInfo
EFI_IPv6_ADDRESS	Destination
IP6_DAD_CALLBACK	Callback
VOID *	Context

Field Documentation

IP6_ADDRESS_INFO* _IP6_DAD_ENTRY::AddressInfo

Referenced by Ip6ConfigOnPolicyChanged(), Ip6FindDADEntry(), Ip6InitDADProcess(), and Ip6NdFasterTimerTicking().

IP6_DAD_CALLBACK _IP6_DAD_ENTRY::Callback

Referenced by Ip6InitDADProcess().

VOID* _IP6_DAD_ENTRY::Context

Referenced by Ip6InitDADProcess().

EFI_IPv6_ADDRESS _IP6_DAD_ENTRY::Destination

Referenced by Ip6InitDADProcess(), and Ip6NdFasterTimerTicking().

LIST_ENTRY _IP6_DAD_ENTRY::Link

Referenced by Ip6InitDADProcess().

UINT32 _IP6_DAD_ENTRY::MaxTransmit

Referenced by Ip6InitDADProcess(), and Ip6NdFasterTimerTicking().

UINT32 _IP6_DAD_ENTRY::Receive

Referenced by Ip6InitDADProcess(), Ip6NdFasterTimerTicking(), and Ip6ProcessNeighborSolicit().

UINT32 _IP6_DAD_ENTRY::RetransTick

Referenced by Ip6InitDADProcess(), and Ip6NdFasterTimerTicking().

UINT32 _IP6_DAD_ENTRY::Signature

Referenced by Ip6InitDADProcess().

UINT32 _IP6_DAD_ENTRY::Transmit

Referenced by Ip6InitDADProcess(), and Ip6NdFasterTimerTicking().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_IP6_DEFAULT_ROUTER Struct Reference

Data Fields

LIST_ENTRY	Link
INTN	RefCnt
UINT16	Lifetime
EFI_IPv6_ADDRESS	Router
IP6_NEIGHBOR_ENTRY *	NeighborCache

Field Documentation

UINT16 _IP6_DEFAULT_ROUTER::Lifetime

Referenced by Ip6CreateDefaultRouter(), Ip6NdTimerTicking(), and Ip6ProcessRouterAdvertise().

LIST_ENTRY _IP6_DEFAULT_ROUTER::Link

Referenced by Ip6CreateDefaultRouter().

IP6_NEIGHBOR_ENTRY* _IP6_DEFAULT_ROUTER::NeighborCache

Referenced by Ip6CreateDefaultRouter(), Ip6CreateNeighborEntry(), Ip6ProcessNeighborAdvertise(), and Ip6ProcessRouterAdvertise().

INTN _IP6_DEFAULT_ROUTER::RefCnt

Referenced by Ip6CreateDefaultRouter().

EFI_IPv6_ADDRESS _IP6_DEFAULT_ROUTER::Router

Referenced by Ip6CreateDefaultRouter(), and Ip6FindDefaultRouter().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_IP6_DELAY_JOIN_LIST Struct Reference

Data Fields

LIST_ENTRY	Link
UINT32	DelayTime in tick per 50 milliseconds
IP6_INTERFACE *	Interface
IP6_ADDRESS_INFO *	AddressInfo
IP6_DAD_CALLBACK	DadCallback
VOID *	Context

Field Documentation

IP6_ADDRESS_INFO* _IP6_DELAY_JOIN_LIST::AddressInfo

Referenced by Ip6ConfigOnPolicyChanged(), Ip6NdFasterTimerTicking(), and Ip6SetAddress().

VOID* _IP6_DELAY_JOIN_LIST::Context

Referenced by Ip6NdFasterTimerTicking(), and Ip6SetAddress().

IP6_DAD_CALLBACK _IP6_DELAY_JOIN_LIST::DadCallback

Referenced by Ip6NdFasterTimerTicking(), and Ip6SetAddress().

UINT32 _IP6_DELAY_JOIN_LIST::DelayTime
in tick per 50 milliseconds

Referenced by Ip6NdFasterTimerTicking(), and Ip6SetAddress().

IP6_INTERFACE* _IP6_DELAY_JOIN_LIST::Interface

Referenced by Ip6NdFasterTimerTicking(), and Ip6SetAddress().

LIST_ENTRY _IP6_DELAY_JOIN_LIST::Link

Referenced by Ip6ConfigOnPolicyChanged(), Ip6NdFasterTimerTicking(), and Ip6SetAddress().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_IP6_ETHE_ADDR_OPTION Struct Reference

Data Fields

UINT8 Type
UINT8 Length
UINT8 EtherAddr [6]

Field Documentation

UINT8 _IP6_ETHE_ADDR_OPTION::EtherAddr[6]

Referenced by Ip6ProcessNeighborAdvertise(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), Ip6SendNeighborAdvertise(), Ip6SendNeighborSolicit(), and Ip6SendRouterSolicit().

UINT8 _IP6_ETHE_ADDR_OPTION::Length

Referenced by Ip6ProcessRedirect(), Ip6SendNeighborAdvertise(), Ip6SendNeighborSolicit(), and Ip6SendRouterSolicit().

UINT8 _IP6_ETHE_ADDR_OPTION::Type

Referenced by Ip6SendNeighborAdvertise(), Ip6SendNeighborSolicit(), and Ip6SendRouterSolicit().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_IP6_FORM_CALLBACK_INFO Struct Reference

Data Fields

UINT32	Signature
EFI_HANDLE	ChildHandle
EFI_HII_CONFIG_ACCESS_PROTOCOL	HiiConfigAccess
EFI_DEVICE_PATH_PROTOCOL *	HiiVendorDevicePath
EFI_HII_HANDLE	RegisteredHandle

Field Documentation

EFI_HANDLE _IP6_FORM_CALLBACK_INFO::ChildHandle

Referenced by Ip6ConfigFormInit(), Ip6ConfigFormUnload(), and Ip6FormExtractConfig().

EFI_HII_CONFIG_ACCESS_PROTOCOL _IP6_FORM_CALLBACK_INFO::HiiConfigAccess

Referenced by Ip6ConfigFormInit(), and Ip6ConfigFormUnload().

EFI_DEVICE_PATH_PROTOCOL* _IP6_FORM_CALLBACK_INFO::HiiVendorDevicePath

Referenced by Ip6ConfigFormInit(), and Ip6ConfigFormUnload().

EFI_HII_HANDLE _IP6_FORM_CALLBACK_INFO::RegisteredHandle

Referenced by Ip6ConfigFormInit(), and Ip6ConfigFormUnload().

UINT32 _IP6_FORM_CALLBACK_INFO::Signature

Referenced by Ip6ConfigFormInit().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_IP6_FRAGMENT_HEADER Struct Reference

Data Fields

UINT8 NextHeader
UINT8 Reserved
UINT16 FragmentOffset
UINT32 Identification

Field Documentation

UINT16 _IP6_FRAGMENT_HEADER::FragmentOffset

Referenced by Ip6FillFragmentHeader(), Ip6IsExtsValid(), and Ip6PreProcessPacket().

UINT32 _IP6_FRAGMENT_HEADER::Identification

Referenced by Ip6FillFragmentHeader(), and Ip6PreProcessPacket().

UINT8 _IP6_FRAGMENT_HEADER::NextHeader

Referenced by Ip6FillFragmentHeader(), and Ip6PreProcessPacket().

UINT8 _IP6_FRAGMENT_HEADER::Reserved

Referenced by Ip6FillFragmentHeader().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_IP6_INTERFACE Struct Reference

Data Fields

UINT32	Signature
LIST_ENTRY	Link
INTN	RefCnt
LIST_ENTRY	AddressList
UINT32	AddressCount
BOOLEAN	Configured
IP6_SERVICE *	Service
EFI_HANDLE	Controller
EFI_HANDLE	Image
LIST_ENTRY	ArpQues
LIST_ENTRY	SentFrames
UINT32	DupAddrDetect
LIST_ENTRY	DupAddrDetectList
LIST_ENTRY	DelayJoinList
LIST_ENTRY	IpInstances
BOOLEAN	PromiscRecv

Field Documentation

UINT32 _IP6_INTERFACE::AddressCount

Referenced by Ip6ConfigOnDhcp6Event(), Ip6ConfigSetMaunualAddress(), Ip6CreateInterface(), and Ip6DestroyPrefixListEntry().

LIST_ENTRY _IP6_INTERFACE::AddressList

Referenced by Ip6CandidateSource(), Ip6ConfigOnDhcp6Event(), Ip6ConfigSetMaunualAddress(), Ip6CreateInterface(), Ip6DestroyPrefixListEntry(), and Ip6IsOneOfSetAddress().

LIST_ENTRY _IP6_INTERFACE::ArpQues

Referenced by Ip6CreateInterface().

BOOLEAN _IP6_INTERFACE::Configured

Referenced by EfiIp6GetModeData(), Ip6CreateInterface(), and Ip6Demultiplex().

EFI_HANDLE _IP6_INTERFACE::Controller

Referenced by Ip6CreateInterface().

LIST_ENTRY _IP6_INTERFACE::DelayJoinList

Referenced by Ip6ConfigOnPolicyChanged(), Ip6CreateInterface(), and Ip6NdFasterTimerTicking().

UINT32 _IP6_INTERFACE::DupAddrDetect

Referenced by Ip6CreateInterface().

LIST_ENTRY _IP6_INTERFACE::DupAddrDetectList

Referenced by Ip6ConfigOnPolicyChanged(), Ip6CreateInterface(), Ip6FindDADEntry(), and Ip6NdFasterTimerTicking().

EFI_HANDLE _IP6_INTERFACE::Image

Referenced by Ip6CreateInterface().

LIST_ENTRY _IP6_INTERFACE::IpInstances

Referenced by Ip6ConfigProtocol(), Ip6CreateInterface(), and Ip6ServiceConfigMnp().

LIST_ENTRY _IP6_INTERFACE::Link

Referenced by Ip6CreateService().

BOOLEAN _IP6_INTERFACE::PromiscRecv

Referenced by Ip6CreateInterface(), and Ip6ServiceConfigMnp().

INTN _IP6_INTERFACE::RefCnt

Referenced by Ip6CreateInterface().

LIST_ENTRY _IP6_INTERFACE::SentFrames

Referenced by Ip6CreateInterface(), and Ip6OnArpResolved().

IP6_SERVICE* _IP6_INTERFACE::Service

Referenced by Ip6CreateInterface(), and Ip6OnArpResolved().

UINT32 _IP6_INTERFACE::Signature

Referenced by Ip6CreateInterface().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_IP6_MTU_OPTION Struct Reference

Data Fields

UINT8 Type
UINT8 Length
UINT16 Reserved
UINT32 Mtu

Field Documentation

UINT8 _IP6_MTU_OPTION::Length

UINT32 _IP6_MTU_OPTION::Mtu

UINT16 _IP6_MTU_OPTION::Reserved

UINT8 _IP6_MTU_OPTION::Type

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_IP6_NEIGHBOR_ENTRY Struct Reference

Data Fields

LIST_ENTRY	Link
LIST_ENTRY	ArpList
INTN	RefCnt
BOOLEAN	IsRouter
BOOLEAN	ArpFree
BOOLEAN	Dynamic
EFI_IPv6_ADDRESS	Neighbor
EFI_MAC_ADDRESS	LinkAddress
EFI_IP6_NEIGHBOR_STATE	State
UINT32	Transmit
UINT32	Ticks
LIST_ENTRY	Frames
IP6_INTERFACE *	Interface
IP6_ARP_CALLBACK	CallBack

Field Documentation

BOOLEAN _IP6_NEIGHBOR_ENTRY::ArpFree

Referenced by Ip6CreateNeighborEntry(), and Ip6SendFrame().

LIST_ENTRY _IP6_NEIGHBOR_ENTRY::ArpList

Referenced by Ip6SendFrame().

IP6_ARP_CALLBACK _IP6_NEIGHBOR_ENTRY::CallBack

Referenced by Ip6CreateNeighborEntry(), Ip6ProcessNeighborAdvertise(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), and Ip6ProcessRouterAdvertise().

BOOLEAN _IP6_NEIGHBOR_ENTRY::Dynamic

Referenced by Ip6AddNeighbor(), Ip6CreateNeighborEntry(), and Ip6NdFasterTimerTicking().

LIST_ENTRY _IP6_NEIGHBOR_ENTRY::Frames

Referenced by Ip6CreateNeighborEntry(), Ip6OnArpResolved(), and Ip6SendFrame().

IP6_INTERFACE* _IP6_NEIGHBOR_ENTRY::Interface

Referenced by Ip6CreateNeighborEntry(), Ip6OnArpResolved(), Ip6Output(), and Ip6SendFrame().

BOOLEAN _IP6_NEIGHBOR_ENTRY::IsRouter

Referenced by Ip6CreateNeighborEntry(), Ip6ProcessNeighborAdvertise(), Ip6ProcessRedirect(), and Ip6ProcessRouterAdvertise().

LIST_ENTRY _IP6_NEIGHBOR_ENTRY::Link

Referenced by Ip6CreateNeighborEntry(), and Ip6DelNeighbor().

EFI_MAC_ADDRESS _IP6_NEIGHBOR_ENTRY::LinkAddress

Referenced by Ip6AddNeighbor(), Ip6BuildEfiNeighborCache(), Ip6CreateNeighborEntry(), Ip6OnArpResolved(), Ip6ProcessNeighborAdvertise(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), Ip6ProcessRouterAdvertise(), and Ip6SendFrame().

EFI_IPv6_ADDRESS _IP6_NEIGHBOR_ENTRY::Neighbor

Referenced by Ip6BuildEfiNeighborCache(), Ip6CreateNeighborEntry(), Ip6FindNeighborEntry(), Ip6NdFasterTimerTicking(), and Ip6Output().

INTN _IP6_NEIGHBOR_ENTRY::RefCnt

Referenced by Ip6CreateNeighborEntry().

EFI_IP6_NEIGHBOR_STATE _IP6_NEIGHBOR_ENTRY::State

Referenced by Ip6AddNeighbor(), Ip6BuildEfiNeighborCache(), Ip6CreateNeighborEntry(), Ip6NdFasterTimerTicking(), Ip6OnArpResolved(), Ip6ProcessNeighborAdvertise(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), Ip6ProcessRouterAdvertise(), and Ip6SendFrame().

UINT32 _IP6_NEIGHBOR_ENTRY::Ticks

Referenced by Ip6AddNeighbor(), Ip6NdFasterTimerTicking(), Ip6OnArpResolved(), Ip6Output(), Ip6ProcessNeighborAdvertise(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), Ip6ProcessRouterAdvertise(), and Ip6SendFrame().

UINT32 _IP6_NEIGHBOR_ENTRY::Transmit

Referenced by Ip6CreateNeighborEntry(), Ip6NdFasterTimerTicking(), and Ip6Output().

- Data Structures
- Files

- Data Structures
- Data Fields

_IP6_PREFIX_INFO_OPTION Struct Reference

Data Fields

UINT8	Type
UINT8	Length
UINT8	PrefixLength
UINT8	Reserved1
UINT32	ValidLifetime
UINT32	PreferredLifetime
UINT32	Reserved2
EFI_IPv6_ADDRESS	Prefix

Field Documentation

UINT8 _IP6_PREFIX_INFO_OPTION::Length

UINT32 _IP6_PREFIX_INFO_OPTION::PreferredLifetime

EFI_IPv6_ADDRESS _IP6_PREFIX_INFO_OPTION::Prefix

UINT8 _IP6_PREFIX_INFO_OPTION::PrefixLength

UINT8 _IP6_PREFIX_INFO_OPTION::Reserved1

UINT32 _IP6_PREFIX_INFO_OPTION::Reserved2

UINT8 _IP6_PREFIX_INFO_OPTION::Type

UINT32 _IP6_PREFIX_INFO_OPTION::ValidLifetime

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_IP6_PREFIX_LIST_ENTRY Struct Reference

Data Fields

LIST_ENTRY	Link
INTN	RefCnt
UINT32	ValidLifetime
UINT32	PreferredLifetime
UINT8	PrefixLength
EFI_IPv6_ADDRESS	Prefix

Field Documentation

LIST_ENTRY _IP6_PREFIX_LIST_ENTRY::Link

Referenced by Ip6CreatePrefixListEntry().

UINT32 _IP6_PREFIX_LIST_ENTRY::PreferredLifetime

Referenced by Ip6CreatePrefixListEntry(), Ip6NdTimerTicking(), Ip6ProcessRouterAdvertise(), and Ip6SelectSourceAddress().

EFI_IPv6_ADDRESS _IP6_PREFIX_LIST_ENTRY::Prefix

Referenced by Ip6BuildPrefixTable(), Ip6CreatePrefixListEntry(), Ip6FindPrefixListEntry(), Ip6IsAnycast(), Ip6SelectSourceAddress(), and Ip6SetAddress().

UINT8 _IP6_PREFIX_LIST_ENTRY::PrefixLength

Referenced by Ip6BuildPrefixTable(), Ip6CreatePrefixListEntry(), Ip6FindPrefixListEntry(), Ip6IsAnycast(), Ip6SelectSourceAddress(), and Ip6SetAddress().

INTN _IP6_PREFIX_LIST_ENTRY::RefCnt

Referenced by Ip6CreatePrefixListEntry().

UINT32 _IP6_PREFIX_LIST_ENTRY::ValidLifetime

Referenced by Ip6CreatePrefixListEntry(), Ip6NdTimerTicking(), and Ip6ProcessRouterAdvertise().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures

- Files

- Data Structures
- Data Fields

_IP6_PROTOCOL Struct Reference

Data Fields

UINT32	Signature
EFI_IP6_PROTOCOL	Ip6Proto
EFI_HANDLE	Handle
INTN	State
IP6_SERVICE *	Service
LIST_ENTRY	Link
UINT8	PrefixLength
NET_MAP	RxTokens
NET_MAP	TxTokens
LIST_ENTRY	Received
LIST_ENTRY	Delivered
EFI_LOCK	RecycleLock
IP6_INTERFACE *	Interface
LIST_ENTRY	AddrLink
EFI_IPv6_ADDRESS *	GroupList
UINT32	GroupCount
EFI_IP6_CONFIG_DATA	ConfigData
BOOLEAN	InDestroy

Field Documentation

LIST_ENTRY _IP6_PROTOCOL::AddrLink

EFI_IP6_CONFIG_DATA _IP6_PROTOCOL::ConfigData

Referenced by EfiIp6Configure(), EfiIp6GetModeData(), EfiIp6Routes(), EfiIp6Transmit(), Ip6DestroyChildEntryByAddr(), Ip6IsValidProtocol(), and Ip6ServiceConfigMnp().

LIST_ENTRY _IP6_PROTOCOL::Delivered

UINT32 _IP6_PROTOCOL::GroupCount

Referenced by EfiIp6GetModeData().

EFI_IPv6_ADDRESS* _IP6_PROTOCOL::GroupList

Referenced by EfiIp6GetModeData().

EFI_HANDLE _IP6_PROTOCOL::Handle

Referenced by Ip6DestroyChildEntryByAddr(), Ip6DestroyChildEntryInHandleBuffer(), and Ip6ServiceBindingCreateChild().

BOOLEAN _IP6_PROTOCOL::InDestroy

Referenced by Ip6ServiceBindingDestroyChild().

IP6_INTERFACE* _IP6_PROTOCOL::Interface

Referenced by EfiIp6GetModeData(), Ip6CancelTxTokens(), and Ip6SentPacketTicking().

EFI_IP6_PROTOCOL _IP6_PROTOCOL::Ip6Proto

Referenced by Ip6ServiceBindingCreateChild(), and Ip6ServiceBindingDestroyChild().

LIST_ENTRY _IP6_PROTOCOL::Link

Referenced by Ip6ServiceBindingCreateChild(), and Ip6ServiceBindingDestroyChild().

UINT8 _IP6_PROTOCOL::PrefixLength

LIST_ENTRY _IP6_PROTOCOL::Received

Referenced by Ip6PacketTimerTicking().

EFI_LOCK _IP6_PROTOCOL::RecycleLock

Referenced by Ip6OnRecyclePacket().

NET_MAP _IP6_PROTOCOL::RxTokens

Referenced by EfiIp6Receive().

IP6_SERVICE* _IP6_PROTOCOL::Service

Referenced by EfiIp6Configure(), EfiIp6GetModeData(), EfiIp6Groups(), EfiIp6Neighbors(), EfiIp6Poll(), EfiIp6Receive(), EfiIp6Routes(), EfiIp6Transmit(), and Ip6ServiceBindingDestroyChild().

UINT32 _IP6_PROTOCOL::Signature

INTN _IP6_PROTOCOL::State

Referenced by EfiIp6Cancel(), EfiIp6Configure(), EfiIp6GetModeData(), EfiIp6Groups(), EfiIp6Neighbors(), EfiIp6Poll(), EfiIp6Receive(), EfiIp6Routes(), EfiIp6Transmit(), Ip6DestroyChildEntryByAddr(), and Ip6IsValidProtocol().

NET_MAP _IP6_PROTOCOL::TxTokens

Referenced by EfiIp6Transmit(), Ip6FreeTxToken(), and Ip6PacketTimerTicking().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_IP6_ROUTE_TABLE Struct Reference

Data Fields

INTN RefCnt
UINT32 TotalNum
LIST_ENTRY RouteArea [IP6_PREFIX_NUM]
IP6_ROUTE_CACHE Cache

Field Documentation

IP6_ROUTE_CACHE _IP6_ROUTE_TABLE::Cache

Referenced by Ip6CreateRouteTable(), and Ip6Route().

INTN _IP6_ROUTE_TABLE::RefCnt

Referenced by Ip6CreateRouteTable().

LIST_ENTRY _IP6_ROUTE_TABLE::RouteArea[IP6_PREFIX_NUM]

Referenced by Ip6CreateRouteTable().

UINT32 _IP6_ROUTE_TABLE::TotalNum

Referenced by Ip6CreateRouteTable().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_IP6_ROUTING_HEADER Struct Reference

Data Fields

UINT8 NextHeader
UINT8 HeaderLen
UINT8 RoutingType
UINT8 SegmentsLeft

Field Documentation

UINT8 _IP6_ROUTING_HEADER::HeaderLen

Referenced by Ip6IsExtsValid().

UINT8 _IP6_ROUTING_HEADER::NextHeader

UINT8 _IP6_ROUTING_HEADER::RoutingType

UINT8 _IP6_ROUTING_HEADER::SegmentsLeft

Referenced by Ip6IsExtsValid().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_IP6_SERVICE Struct Reference

Data Fields

UINT32	Signature
EFI_SERVICE_BINDING_PROTOCOL	ServiceBinding
INTN	State
UINTN	NumChildren
LIST_ENTRY	Children
LIST_ENTRY	Interfaces
IP6_INTERFACE *	DefaultInterface
IP6_ROUTE_TABLE *	RouteTable
IP6_LINK_RX_TOKEN	RecvRequest
IP6_ASSEMBLE_TABLE	Assemble
IP6_MLD_SERVICE_DATA	MldCtrl
EFI_IPv6_ADDRESS	LinkLocalAddr
BOOLEAN	LinkLocalOk
BOOLEAN	LinkLocalDadFail
BOOLEAN	Dhcp6NeedStart
BOOLEAN	Dhcp6NeedInfoRequest
UINT8	CurHopLimit
UINT32	LinkMTU
UINT32	BaseReachableTime
UINT32	ReachableTime
UINT32	RetransTimer
LIST_ENTRY	NeighborTable
LIST_ENTRY	OnlinkPrefix
LIST_ENTRY	AutonomousPrefix
LIST_ENTRY	DefaultRouterList
UINT32	RoundRobin
UINT8	InterfaceIdLen
UINT8 *	InterfaceId
BOOLEAN	RouterAdvertiseReceived
UINT8	SolicitTimer
UINT32	Ticks
EFI_HANDLE	Image
EFI_HANDLE	Controller
EFI_HANDLE	MnpChildHandle
EFI_MANAGED_NETWORK_PROTOCOL *	Mnp
EFI_MANAGED_NETWORK_CONFIG_DATA	MnpConfigData
EFI_SIMPLE_NETWORK_MODE	SnpMode
EFI_EVENT	Timer
EFI_EVENT	FasterTimer
IP6_CONFIG_INSTANCE	Ip6ConfigInstance
CHAR16 *	MacString
UINT32	MaxPacketSize
UINT32	OldMaxPacketSize

Field Documentation

IP6_ASSEMBLE_TABLE _IP6_SERVICE::Assemble

Referenced by Ip6CreateService().

LIST_ENTRY _IP6_SERVICE::AutonomousPrefix

Referenced by Ip6CreateService(), and Ip6SetAddress().

UINT32 _IP6_SERVICE::BaseReachableTime

Referenced by Ip6CreateService().

LIST_ENTRY _IP6_SERVICE::Children

Referenced by Ip6CreateService(), Ip6DriverBindingStop(), and Ip6ServiceBindingCreateChild().

EFI_HANDLE _IP6_SERVICE::Controller

Referenced by Ip6ConfigDestroyDhcp6(), Ip6ConfigFormInit(), Ip6ConfigFormUnload(), Ip6ConfigStartStatefulAutoConfig(), and Ip6CreateService().

UINT8 _IP6_SERVICE::CurHopLimit

Referenced by Ip6CreateService().

IP6_INTERFACE* _IP6_SERVICE::DefaultInterface

Referenced by Ip6ConfigOnDhcp6Event(), Ip6ConfigProtocol(), Ip6ConfigSetMaunualAddress(), and Ip6CreateService().

LIST_ENTRY _IP6_SERVICE::DefaultRouterList

Referenced by Ip6CreateService().

BOOLEAN _IP6_SERVICE::Dhcp6NeedInfoRequest

Referenced by Ip6ConfigStartStatefulAutoConfig(), Ip6CreateService(), and Ip6OnDADFinished().

BOOLEAN _IP6_SERVICE::Dhcp6NeedStart

Referenced by Ip6ConfigStartStatefulAutoConfig(), Ip6CreateService(), and Ip6OnDADFinished().

EFI_EVENT _IP6_SERVICE::FasterTimer

Referenced by Ip6CreateService(), Ip6DriverBindingStart(), and Ip6OnDADFinished().

EFI_HANDLE _IP6_SERVICE::Image

Referenced by Ip6ConfigDestroyDhcp6(), Ip6ConfigFormInit(), Ip6ConfigFormUnload(), Ip6ConfigStartStatefulAutoConfig(), and Ip6CreateService().

UINT8* _IP6_SERVICE::InterfaceId

Referenced by Ip6CreateService().

UINT8 _IP6_SERVICE::InterfaceIdLen

Referenced by Ip6CreateService().

LIST_ENTRY _IP6_SERVICE::Interfaces

Referenced by Ip6ConfigSetMaunualAddress(), Ip6CreateService(), and Ip6NdFasterTimerTicking().

IP6_CONFIG_INSTANCE _IP6_SERVICE::Ip6ConfigInstance

Referenced by Ip6CreateService(), Ip6DriverBindingStart(), Ip6DriverBindingStop(), Ip6InitDADProcess(), Ip6NdFasterTimerTicking(), and Ip6OnDADFinished().

EFI_IPv6_ADDRESS _IP6_SERVICE::LinkLocalAddr

Referenced by Ip6CreateService(), and Ip6OnDADFinished().

BOOLEAN _IP6_SERVICE::LinkLocalDadFail

Referenced by EfiIp6ConfigSetData(), EfiIp6Configure(), EfiIp6GetModeData(), EfiIp6Groups(), EfiIp6Neighbors(), EfiIp6Poll(), EfiIp6Receive(), EfiIp6Routes(), EfiIp6Transmit(), Ip6CreateService(), Ip6OnDADFinished(), and Ip6ServiceBindingCreateChild().

BOOLEAN _IP6_SERVICE::LinkLocalOk

Referenced by Ip6ConfigStartStatefulAutoConfig(), Ip6CreateService(), Ip6OnDADFinished(), and Ip6SendFrame().

UINT32 _IP6_SERVICE::LinkMTU

Referenced by Ip6CreateService().

CHAR16* _IP6_SERVICE::MacString

Referenced by EfiIp6ConfigSetData(), Ip6ConfigInitInstance(), Ip6CreateService(), and Ip6ManualAddrDadCallback().

UINT32 _IP6_SERVICE::MaxPacketSize

Referenced by EfiIp6GetModeData(), and Ip6CreateService().

IP6_MLD_SERVICE_DATA _IP6_SERVICE::MldCtrl

Referenced by Ip6CreateService().

EFI_MANAGED_NETWORK_PROTOCOL* _IP6_SERVICE::Mnp

Referenced by EfiIp6GetModeData(), EfiIp6Poll(), Ip6CancelFrames(), Ip6CreateService(), Ip6OnArpResolved(), Ip6OnDADFinished(), and Ip6SendFrame().

EFI_HANDLE _IP6_SERVICE::MnpChildHandle

Referenced by Ip6CreateService(), Ip6ServiceBindingCreateChild(), and Ip6ServiceBindingDestroyChild().

EFI_MANAGED_NETWORK_CONFIG_DATA _IP6_SERVICE::MnpConfigData

Referenced by Ip6CreateService().

LIST_ENTRY _IP6_SERVICE::NeighborTable

Referenced by Ip6BuildEfiNeighborCache(), Ip6CreateService(), and Ip6NdFasterTimerTicking().

UINTN _IP6_SERVICE::NumChildren

Referenced by Ip6CreateService(), Ip6ServiceBindingCreateChild(), and Ip6ServiceBindingDestroyChild().

UINT32 _IP6_SERVICE::OldMaxPacketSize

Referenced by Ip6CreateService().

LIST_ENTRY _IP6_SERVICE::OnlinkPrefix

Referenced by Ip6BuildPrefixTable(), Ip6CreateService(), and Ip6SetAddress().

UINT32 _IP6_SERVICE::ReachableTime

IP6_LINK_RX_TOKEN _IP6_SERVICE::RecvRequest

Referenced by Ip6CreateService().

UINT32 _IP6_SERVICE::RetransTimer

Referenced by Ip6CreateService(), and Ip6NdFasterTimerTicking().

UINT32 _IP6_SERVICE::RoundRobin

Referenced by Ip6CreateService().

BOOLEAN _IP6_SERVICE::RouterAdvertiseReceived

Referenced by Ip6CreateService(), and Ip6NdFasterTimerTicking().

IP6_ROUTE_TABLE* _IP6_SERVICE::RouteTable

Referenced by EfiIp6GetModeData(), EfiIp6Routes(), Ip6ConfigGetIfInfo(), and Ip6CreateService().

EFI_SERVICE_BINDING_PROTOCOL _IP6_SERVICE::ServiceBinding

Referenced by Ip6CreateService(), and Ip6DriverBindingStart().

UINT32 _IP6_SERVICE::Signature

Referenced by Ip6CreateService(), and Ip6OnArpResolved().

EFI_SIMPLE_NETWORK_MODE _IP6_SERVICE::SnpMode

Referenced by Ip6ConfigInitInstance(), Ip6CreateService(), Ip6NdFasterTimerTicking(), Ip6OnArpResolved(), and Ip6SendFrame().

UINT8 _IP6_SERVICE::SolicitTimer

Referenced by Ip6CreateService(), and Ip6NdFasterTimerTicking().

INTN _IP6_SERVICE::State

Referenced by Ip6AcceptFrame(), Ip6CreateService(), and Ip6DriverBindingStop().

UINT32 _IP6_SERVICE::Ticks

Referenced by Ip6CreateService(), and Ip6NdFasterTimerTicking().

EFI_EVENT _IP6_SERVICE::Timer

Referenced by Ip6CreateService(), Ip6DriverBindingStart(), and Ip6OnDADFinished().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_IPSEC_PAD_ENTRY Struct Reference

Data Fields

EFI_IPSEC_PAD_ID * Id
EFI_IPSEC_PAD_DATA * Data
LIST_ENTRY List

Field Documentation

EFI_IPSEC_PAD_DATA* _IPSEC_PAD_ENTRY::Data

Referenced by GetPadEntry(), IkeNegotiate(), Ikev2AuthCertGenerator(), Ikev2HandleSa(), Ikev2NegotiateSa(), and SetPadEntry().

EFI_IPSEC_PAD_ID* _IPSEC_PAD_ENTRY::Id

Referenced by GetPadEntry(), IpSecLookupPadEntry(), and SetPadEntry().

LIST_ENTRY _IPSEC_PAD_ENTRY::List

Referenced by SetPadEntry().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_IPSEC_PRIVATE_DATA Struct Reference

Data Fields

UINT32	Signature
EFI_HANDLE	Handle
EFI_HANDLE	ImageHandle
EFI_IPSEC2_PROTOCOL	IpSec
EFI_IPSEC_CONFIG_PROTOCOL	IpSecConfig
BOOLEAN	SetBySelf
LIST_ENTRY	Udp4List
UINTN	Udp4Num
LIST_ENTRY	Udp6List
UINTN	Udp6Num
LIST_ENTRY	Ikev1SessionList
LIST_ENTRY	Ikev1EstablishedList
LIST_ENTRY	Ikev2SessionList
LIST_ENTRY	Ikev2EstablishedList
BOOLEAN	IsIPsecDisabling

Field Documentation

EFI_HANDLE _IPSEC_PRIVATE_DATA::Handle

Referenced by IpSecDriverEntryPoint().

LIST_ENTRY _IPSEC_PRIVATE_DATA::Ikev1EstablishedList

Referenced by Ikev2InfoParser(), Ikev2OnPacketSent(), Ikev2ResendNotify(), and IpSecDriverEntryPoint().

LIST_ENTRY _IPSEC_PRIVATE_DATA::Ikev1SessionList

Referenced by IpSecDriverEntryPoint().

LIST_ENTRY _IPSEC_PRIVATE_DATA::Ikev2EstablishedList

Referenced by IkeNegotiate(), Ikev2HandleChildSa(), Ikev2HandleInfo(), Ikev2InfoParser(), Ikev2OnPacketSent(), Ikev2ResendNotify(), IpSecDriverEntryPoint(), and IpSecStop().

LIST_ENTRY _IPSEC_PRIVATE_DATA::Ikev2SessionList

Referenced by Ikev2HandleSa(), Ikev2NegotiateSa(), Ikev2ResendNotify(), IpSecDriverEntryPoint(), and IpSecStop().

EFI_HANDLE _IPSEC_PRIVATE_DATA::ImageHandle

Referenced by IpSecDriverEntryPoint().

EFI_IPSEC2_PROTOCOL _IPSEC_PRIVATE_DATA::IpSec

Referenced by IkeDispatch(), Ikev2InfoParser(), Ikev2OnPacketSent(), Ikev2ResendNotify(), and IpSecDriverEntryPoint().

EFI_IPSEC_CONFIG_PROTOCOL _IPSEC_PRIVATE_DATA::IpSecConfig

Referenced by Ikev2ChildSaSilentDelete(), Ikev2StoreSaData(), and IpSecDriverEntryPoint().

BOOLEAN _IPSEC_PRIVATE_DATA::IsIPsecDisabling

Referenced by Ikev2InfoParser(), Ikev2OnPacketSent(), Ikev2ResendNotify(), and IpSecCleanupAllSa().

BOOLEAN _IPSEC_PRIVATE_DATA::SetBySelf

UINT32 _IPSEC_PRIVATE_DATA::Signature

Referenced by IpSecDriverEntryPoint().

LIST_ENTRY _IPSEC_PRIVATE_DATA::Udp4List

Referenced by IpSecDriverEntryPoint(), and IpSecStop().

UINTN _IPSEC_PRIVATE_DATA::Udp4Num

Referenced by IpSecStop().

LIST_ENTRY _IPSEC_PRIVATE_DATA::Udp6List

Referenced by IpSecDriverEntryPoint(), and IpSecStop().

UINTN _IPSEC_PRIVATE_DATA::Udp6Num

Referenced by IpSecStop().

- Data Structures
- Data Fields

_IPSEC_RECYCLE_CONTEXT Struct Reference

Data Fields

EFI_IPSEC_FRAGMENT_DATA * FragmentTable
UINT8 * PayloadBuffer

Field Documentation

EFI_IPSEC_FRAGMENT_DATA* _IPSEC_RECYCLE_CONTEXT::FragmentTable

Referenced by IpSecEspInboundPacket(), IpSecEspOutboundPacket(), and IpSecRecycleCallback().

UINT8* _IPSEC_RECYCLE_CONTEXT::PayloadBuffer

Referenced by IpSecEspInboundPacket(), IpSecEspOutboundPacket(), and IpSecRecycleCallback().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_IPSEC_SAD_DATA Struct Reference

Data Fields

EFI_IPSEC_MODE	Mode
UINT64	SequenceNumber
UINT8	AntiReplayWindowSize
UINT64	AntiReplayBitmap [4]
EFI_IPSEC_ALGO_INFO	AlgoInfo
EFI_IPSEC_SA_LIFETIME	SaLifetime
UINT32	PathMTU
IPSEC_SPD_ENTRY *	SpdEntry
EFI_IPSEC_SPD_SELECTOR *	SpdSelector
BOOLEAN	ESNEnabled
BOOLEAN	ManualSet
EFI_IP_ADDRESS	TunnelDestAddress
EFI_IP_ADDRESS	TunnelSourceAddress

Field Documentation

EFI_IPSEC_ALGO_INFO _IPSEC_SAD_DATA::AlgoInfo

Referenced by GetSadEntry(), IpSecEspInboundPacket(), IpSecEspOutboundPacket(), and SetSadEntry().

UINT64 _IPSEC_SAD_DATA::AntiReplayBitmap[4]

Referenced by SetSadEntry().

UINT8 _IPSEC_SAD_DATA::AntiReplayWindowSize

Referenced by GetSadEntry(), and SetSadEntry().

BOOLEAN _IPSEC_SAD_DATA::ESNEnabled

Referenced by IpSecLookupSadEntry(), and SetSadEntry().

BOOLEAN _IPSEC_SAD_DATA::ManualSet

Referenced by GetSadEntry(), IpSecEspInboundPacket(), IpSecEspOutboundPacket(), IpSecLookupSadEntry(), and SetSadEntry().

EFI_IPSEC_MODE _IPSEC_SAD_DATA::Mode

Referenced by GetSadEntry(), IpSecEspInboundPacket(), IpSecEspOutboundPacket(), IpSecLookupSadBySpi(), and SetSadEntry().

UINT32 _IPSEC_SAD_DATA::PathMTU

Referenced by GetSadEntry(), and SetSadEntry().

EFI_IPSEC_SA_LIFETIME _IPSEC_SAD_DATA::SaLifetime

Referenced by GetSadEntry(), and SetSadEntry().

UINT64 _IPSEC_SAD_DATA::SequenceNumber

Referenced by GetSadEntry(), IpSecEspOutboundPacket(), IpSecLookupSadEntry(), and SetSadEntry().

IPSEC_SPD_ENTRY* _IPSEC_SAD_DATA::SpdEntry

Referenced by SetSadEntry(), and SetSpdEntry().

EFI_IPSEC_SPD_SELECTOR* _IPSEC_SAD_DATA::SpdSelector

Referenced by GetSadEntry(), IpSecEspInboundPacket(), IpSecLookupSadBySpd(), IpSecLookupSadBySpi(), and SetSadEntry().

EFI_IP_ADDRESS _IPSEC_SAD_DATA::TunnelDestAddress

Referenced by GetSadEntry(), IpSecEspOutboundPacket(), IpSecLookupSadBySpi(), and SetSadEntry().

EFI_IP_ADDRESS _IPSEC_SAD_DATA::TunnelSourceAddress

Referenced by GetSadEntry(), IpSecEspOutboundPacket(), and SetSadEntry().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_IPSEC_SAD_ENTRY Struct Reference

Data Fields

EFI_IPSEC_SA_ID * Id
IPSEC_SAD_DATA * Data
LIST_ENTRY List
LIST_ENTRY BySpd

Field Documentation

LIST_ENTRY _IPSEC_SAD_ENTRY::BySpd

Referenced by SetSadEntry(), and SetSpdEntry().

IPSEC_SAD_DATA* _IPSEC_SAD_ENTRY::Data

Referenced by GetSadEntry(), IpSecEspInboundPacket(), IpSecLookupSadBySpd(), IpSecLookupSadBySpi(), IpSecLookupSadEntry(), SetSadEntry(), and SetSpdEntry().

EFI_IPSEC_SA_ID* _IPSEC_SAD_ENTRY::Id

Referenced by GetSadEntry(), GetSpdEntry(), IpSecEspInboundPacket(), IpSecLookupSadBySpi(), SetSadEntry(), and SetSpdEntry().

LIST_ENTRY _IPSEC_SAD_ENTRY::List

Referenced by SetSadEntry().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_IPSEC_SPD_DATA Struct Reference

Data Fields

CHAR16	Name [100]
UINT32	PackageFlag
EFI_IPSEC_TRAFFIC_DIR	TrafficDirection
EFI_IPSEC_ACTION	Action
EFI_IPSEC_PROCESS_POLICY *	ProcessingPolicy
LIST_ENTRY	Sas

Field Documentation

EFI_IPSEC_ACTION _IPSEC_SPD_DATA::Action

Referenced by GetSpdEntry(), SetSadEntry(), and SetSpdEntry().

CHAR16 _IPSEC_SPD_DATA::Name[100]

Referenced by GetSpdEntry(), and SetSpdEntry().

UINT32 _IPSEC_SPD_DATA::PackageFlag

Referenced by GetSpdEntry(), and SetSpdEntry().

EFI_IPSEC_PROCESS_POLICY* _IPSEC_SPD_DATA::ProcessingPolicy

Referenced by GetSpdEntry(), Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), and SetSpdEntry().

LIST_ENTRY _IPSEC_SPD_DATA::Sas

Referenced by GetSpdEntry(), SetSadEntry(), and SetSpdEntry().

EFI_IPSEC_TRAFFIC_DIR _IPSEC_SPD_DATA::TrafficDirection

Referenced by GetSpdEntry(), and SetSpdEntry().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_IPSEC_SPD_ENTRY Struct Reference

Data Fields

EFI_IPSEC_SPD_SELECTOR * Selector
IPSEC_SPD_DATA * Data
LIST_ENTRY List

Field Documentation

IPSEC_SPD_DATA* _IPSEC_SPD_ENTRY::Data

Referenced by GetSpdEntry(), Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), SetSadEntry(), and SetSpdEntry().

LIST_ENTRY _IPSEC_SPD_ENTRY::List

Referenced by SetSpdEntry().

EFI_IPSEC_SPD_SELECTOR* _IPSEC_SPD_ENTRY::Selector

Referenced by GetSpdEntry(), IkeSearchSpdEntry(), IpSecProcess(), SetSadEntry(), and SetSpdEntry().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_ISCSI_ADDTIONAL_HEADER Struct Reference

Data Fields

UINT16 Length
UINT8 Type
UINT8 TypeSpecific [1]

Field Documentation

UINT16 _ISCSI_ADDTIONAL_HEADER::Length

Referenced by IScsiNewScsiCmdPdu().

UINT8 _ISCSI_ADDTIONAL_HEADER::Type

Referenced by IScsiNewScsiCmdPdu().

UINT8 _ISCSI_ADDTIONAL_HEADER::TypeSpecific[1]

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_ISCSI_ASYNC_MESSAGE Struct Reference

Data Fields

UINT8 OpCode
UINT8 Reserved1 [8]
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT8 Lun [8]
UINT32 InitiatorTaskTag
UINT32 Reserved2
UINT32 StatSN
UINT32 ExpCmdSN
UINT32 MaxCmdSN
UINT8 AsyncEvent
UINT8 AsyncVCode
UINT16 Parameter1
UINT16 Parameter2
UINT16 Parameter3
UINT32 Reserved3

Field Documentation

UINT8 _ISCSI_ASYNC_MESSAGE::AsyncEvent

UINT8 _ISCSI_ASYNC_MESSAGE::AsyncVCode

UINT8 _ISCSI_ASYNC_MESSAGE::DataSegmentLength[3]

UINT32 _ISCSI_ASYNC_MESSAGE::ExpCmdSN

UINT32 _ISCSI_ASYNC_MESSAGE::InitiatorTaskTag

UINT8 _ISCSI_ASYNC_MESSAGE::Lun[8]

UINT32 _ISCSI_ASYNC_MESSAGE::MaxCmdSN

UINT8 _ISCSI_ASYNC_MESSAGE::OpCode

UINT16 _ISCSI_ASYNC_MESSAGE::Parameter1

UINT16 _ISCSI_ASYNC_MESSAGE::Parameter2

UINT16 _ISCSI_ASYNC_MESSAGE::Parameter3

UINT8 _ISCSI_ASYNC_MESSAGE::Reserved1[8]

UINT32 _ISCSI_ASYNC_MESSAGE::Reserved2

UINT32 _ISCSI_ASYNC_MESSAGE::Reserved3

UINT32 _ISCSI_ASYNC_MESSAGE::StatSN

UINT8 _ISCSI_ASYNC_MESSAGE::TotalAHSLength

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_ISCSI_ATTEMPT_CONFIG_NVDATA Struct Reference

Data Fields

```
                                    LIST_ENTRY Link
                                         UINT8 NicIndex
                                         UINT8 AttemptConfigIndex
                                       BOOLEAN DhcpSuccess
                                       BOOLEAN ValidiBFTPath
                                       BOOLEAN ValidPath
                                         UINT8 AutoConfigureMode
                                   EFI_STRING_ID AttemptTitleToken
                                   EFI_STRING_ID AttemptTitleHelpToken
                                        CHAR8 AttemptName [ATTEMPT_NAME_MAX_SIZE]
                                        CHAR8 MacString [ISCSI_MAX_MAC_STRING_LEN]
                                EFI_IP_ADDRESS PrimaryDns
                                EFI_IP_ADDRESS SecondaryDns
                                EFI_IP_ADDRESS DhcpServer
                ISCSI_SESSION_CONFIG_NVDATA SessionConfigData
                                         UINT8 AuthenticationType
union {
  ISCSI_CHAP_AUTH_CONFIG_NVDATA  CHAP
}                                              AuthConfigData
                                       BOOLEAN AutoConfigureSuccess
```

Field Documentation

UINT8 _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptConfigIndex

Referenced by IScsiConfigDeleteAttempts(), IScsiConfigDisplayOrderAttempts(), IScsiConfigGetAttemptByConfigIndex(), IScsiConfigProcessDefault(), IScsiConfigUpdateAttempt(), IScsiFillNICAndTargetSections(), IScsiGetConfigData(), and IScsiStart().

CHAR8 _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptName[ATTEMPT_NAME_MAX_SIZE]

Referenced by IScsiConfigProcessDefault(), IScsiConfigUpdateAttempt(), IScsiConvertIfrNvDataToAttemptConfigData(), and IScsiFormCallback().

EFI_STRING_ID _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptTitleHelpToken

Referenced by IScsiConfigDisplayDeleteAttempts(), IScsiConfigProcessDefault(), and IScsiConfigUpdateAttempt().

EFI_STRING_ID _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptTitleToken

Referenced by IScsiConfigDisplayDeleteAttempts(), IScsiConfigDisplayOrderAttempts(), and IScsiConfigUpdateAttempt().

union { ... } _ISCSI_ATTEMPT_CONFIG_NVDATA::AuthConfigData

Referenced by IScsiConfigProcessDefault(), IScsiFillNICAndTargetSections(), IScsiFormCallback(), and IScsiStart().

UINT8 _ISCSI_ATTEMPT_CONFIG_NVDATA::AuthenticationType

Referenced by IScsiConfigProcessDefault(), IScsiFillNICAndTargetSections(), IScsiRemoveNic(), and IScsiStart().

UINT8 _ISCSI_ATTEMPT_CONFIG_NVDATA::AutoConfigureMode

Referenced by IScsiFormCallback(), IScsiGetConfigData(), and IScsiStart().

BOOLEAN _ISCSI_ATTEMPT_CONFIG_NVDATA::AutoConfigureSuccess

Referenced by IScsiGetConfigData(), and IScsiStart().

ISCSI_CHAP_AUTH_CONFIG_NVDATA _ISCSI_ATTEMPT_CONFIG_NVDATA::CHAP

Referenced by IScsiConfigProcessDefault(), IScsiFillNICAndTargetSections(), IScsiFormCallback(), and IScsiStart().

EFI_IP_ADDRESS _ISCSI_ATTEMPT_CONFIG_NVDATA::DhcpServer

Referenced by IScsiDhcp6ParseReply(), and IScsiFillNICAndTargetSections().

BOOLEAN _ISCSI_ATTEMPT_CONFIG_NVDATA::DhcpSuccess

Referenced by IScsiGetConfigData(), and IScsiStart().

LIST_ENTRY _ISCSI_ATTEMPT_CONFIG_NVDATA::Link

Referenced by IScsiConfigDeleteAttempts(), IScsiConfigOrderAttempts(), IScsiGetConfigData(), and IScsiRemoveNic().

CHAR8 _ISCSI_ATTEMPT_CONFIG_NVDATA::MacString[ISCSI_MAX_MAC_STRING_LEN]

Referenced by IScsiConfigDeleteAttempts(), IScsiConfigProcessDefault(), IScsiGetConfigData(), and IScsiStart().

UINT8 _ISCSI_ATTEMPT_CONFIG_NVDATA::NicIndex

Referenced by IScsiConfigGetAttemptByNic(), IScsiConfigProcessDefault(), IScsiFillNICAndTargetSections(), IScsiRemoveNic(), IScsiStart(), and UpdateName().

EFI_IP_ADDRESS _ISCSI_ATTEMPT_CONFIG_NVDATA::PrimaryDns

Referenced by IScsiDhcp6ParseReply(), and IScsiFillNICAndTargetSections().

EFI_IP_ADDRESS _ISCSI_ATTEMPT_CONFIG_NVDATA::SecondaryDns

Referenced by IScsiDhcp6ParseReply(), and IScsiFillNICAndTargetSections().

ISCSI_SESSION_CONFIG_NVDATA _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData

Referenced by IScsiCHAPToSendReq(), IScsiConfigDeleteAttempts(), IScsiConfigGetAttemptByNic(), IScsiConfigProcessDefault(), IScsiExtScsiPassThruBuildDevicePath(), IScsiExtScsiPassThruGetNextTargetLun(), IScsiExtScsiPassThruGetTargetLun(), IScsiFillNICAndTargetSections(), IScsiFormCallback(), IScsiGetConfigData(), IScsiRemoveNic(), and IScsiStart().

BOOLEAN _ISCSI_ATTEMPT_CONFIG_NVDATA::ValidiBFTPath

Referenced by IScsiFillNICAndTargetSections(), and IScsiStart().

BOOLEAN _ISCSI_ATTEMPT_CONFIG_NVDATA::ValidPath

Referenced by IScsiGetConfigData(), and IScsiStart().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_BASIC_HEADER Struct Reference

Data Fields

UINT8 OpCode
UINT8 Flags
UINT16 OpCodeSpecific1
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT8 Lun [8]
UINT32 InitiatorTaskTag
UINT32 OpCodeSpecific2 [7]

Detailed Description

iSCSI Basic Header Segment

Field Documentation

UINT8 _ISCSI_BASIC_HEADER::DataSegmentLength[3]

UINT8 _ISCSI_BASIC_HEADER::Flags

UINT32 _ISCSI_BASIC_HEADER::InitiatorTaskTag

UINT8 _ISCSI_BASIC_HEADER::Lun[8]

UINT8 _ISCSI_BASIC_HEADER::OpCode

UINT16 _ISCSI_BASIC_HEADER::OpCodeSpecific1

UINT32 _ISCSI_BASIC_HEADER::OpCodeSpecific2[7]

UINT8 _ISCSI_BASIC_HEADER::TotalAHSLength

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_BI_EXP_READ_DATA_LEN_AHS Struct Reference

Data Fields

UINT16 Length
UINT8 Type
UINT8 Reserved
UINT32 ExpReadDataLength

Field Documentation

UINT32 _ISCSI_BI_EXP_READ_DATA_LEN_AHS::ExpReadDataLength

Referenced by IScsiNewScsiCmdPdu().

UINT16 _ISCSI_BI_EXP_READ_DATA_LEN_AHS::Length

Referenced by IScsiNewScsiCmdPdu().

UINT8 _ISCSI_BI_EXP_READ_DATA_LEN_AHS::Reserved

UINT8 _ISCSI_BI_EXP_READ_DATA_LEN_AHS::Type

Referenced by IScsiNewScsiCmdPdu().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_CHAP_AUTH_CONFIG_NVDATA Struct Reference

Data Fields

UINT8 CHAPType
CHAR8 CHAPName [ISCSI_CHAP_NAME_STORAGE]
CHAR8 CHAPSecret [ISCSI_CHAP_SECRET_STORAGE]
CHAR8 ReverseCHAPName [ISCSI_CHAP_NAME_STORAGE]
CHAR8 ReverseCHAPSecret [ISCSI_CHAP_SECRET_STORAGE]

Field Documentation

CHAR8 _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPName[ISCSI_CHAP_NAME_STORAGE]

Referenced by IScsiCHAPToSendReq(), IScsiConvertAttemptConfigDataToIfrNvData(), IScsiFillNICAndTargetSections(), and IScsiFormCallback().

CHAR8 _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPSecret[ISCSI_CHAP_SECRET_STORAGE]

Referenced by IScsiCHAPOnRspReceived(), IScsiConvertAttemptConfigDataToIfrNvData(), IScsiFillNICAndTargetSections(), and IScsiFormCallback().

UINT8 _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPType

Referenced by IScsiCHAPOnRspReceived(), IScsiCHAPToSendReq(), IScsiConfigProcessDefault(), IScsiConvertAttemptConfigDataToIfrNvData(), IScsiExtScsiPassThruBuildDevicePath(), and IScsiFillNICAndTargetSections().

CHAR8 _ISCSI_CHAP_AUTH_CONFIG_NVDATA::ReverseCHAPName[ISCSI_CHAP_NAME_STORAGE]

Referenced by IScsiConvertAttemptConfigDataToIfrNvData(), IScsiFillNICAndTargetSections(), and IScsiFormCallback().

CHAR8 _ISCSI_CHAP_AUTH_CONFIG_NVDATA::ReverseCHAPSecret[ISCSI_CHAP_SECRET_STORAGE]

Referenced by IScsiConvertAttemptConfigDataToIfrNvData(), IScsiFillNICAndTargetSections(), and IScsiFormCallback().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures

- Files

- Data Structures
- Data Fields

_ISCSI_CHAP_AUTH_DATA Struct Reference

Data Fields

ISCSI_CHAP_AUTH_CONFIG_NVDATA * AuthConfig
UINT32 InIdentifier
UINT8 InChallenge [ISCSI_CHAP_AUTH_MAX_LEN]
UINT32 InChallengeLength
UINT8 CHAPResponse [ISCSI_CHAP_RSP_LEN]
UINT32 OutIdentifier
UINT8 OutChallenge [ISCSI_CHAP_AUTH_MAX_LEN]
UINT32 OutChallengeLength

Detailed Description

ISCSI CHAP Authentication Data

Field Documentation

ISCSI_CHAP_AUTH_CONFIG_NVDATA* _ISCSI_CHAP_AUTH_DATA::AuthConfig

Referenced by IScsiCHAPOnRspReceived(), IScsiCHAPToSendReq(), IScsiExtScsiPassThruBuildDevicePath(), and IScsiStart().

UINT8 _ISCSI_CHAP_AUTH_DATA::CHAPResponse[ISCSI_CHAP_RSP_LEN]

Referenced by IScsiCHAPOnRspReceived(), and IScsiCHAPToSendReq().

UINT8 _ISCSI_CHAP_AUTH_DATA::InChallenge[ISCSI_CHAP_AUTH_MAX_LEN]

Referenced by IScsiCHAPOnRspReceived().

UINT32 _ISCSI_CHAP_AUTH_DATA::InChallengeLength

Referenced by IScsiCHAPOnRspReceived().

UINT32 _ISCSI_CHAP_AUTH_DATA::InIdentifier

Referenced by IScsiCHAPOnRspReceived().

UINT8 _ISCSI_CHAP_AUTH_DATA::OutChallenge[ISCSI_CHAP_AUTH_MAX_LEN]

Referenced by IScsiCHAPToSendReq().

UINT32 _ISCSI_CHAP_AUTH_DATA::OutChallengeLength

Referenced by IScsiCHAPToSendReq().

UINT32 _ISCSI_CHAP_AUTH_DATA::OutIdentifier

Referenced by IScsiCHAPToSendReq().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_CONFIG_IFR_NVDATA Struct Reference

Data Fields

CHAR16 InitiatorName [ISCSI_NAME_MAX_SIZE]
CHAR16 AttemptName [ATTEMPT_NAME_MAX_SIZE]
UINT8 Enabled
UINT8 IpMode
UINT8 ConnectRetryCount
UINT8 Padding1
UINT16 ConnectTimeout
UINT8 InitiatorInfoFromDhcp
UINT8 TargetInfoFromDhcp
CHAR16 LocalIp [IP4_STR_MAX_SIZE]
CHAR16 SubnetMask [IP4_STR_MAX_SIZE]
CHAR16 Gateway [IP4_STR_MAX_SIZE]
CHAR16 TargetName [ISCSI_NAME_MAX_SIZE]
CHAR16 TargetIp [IP_STR_MAX_SIZE]
UINT16 TargetPort
CHAR16 BootLun [ISCSI_LUN_STR_MAX_LEN]
UINT8 AuthenticationType
UINT8 CHAPType
CHAR16 CHAPName [ISCSI_CHAP_NAME_STORAGE]
CHAR16 CHAPSecret [ISCSI_CHAP_SECRET_STORAGE]
CHAR16 ReverseCHAPName [ISCSI_CHAP_NAME_STORAGE]
CHAR16 ReverseCHAPSecret [ISCSI_CHAP_SECRET_STORAGE]
BOOLEAN MutualRequired
UINT8 Padding2
CHAR16 KerberosUserName [KERBEROS_NAME_MAX_LEN]
CHAR16 KerberosUserSecret [KERBEROS_SECRET_STORAGE]
CHAR16 KerberosKDCName [KERBEROS_NAME_MAX_LEN]
CHAR16 KerberosKDCRealm [KERBEROS_NAME_MAX_LEN]
CHAR16 KerberosKDCIp [IP_STR_MAX_SIZE]
UINT16 KerberosKDCPort
UINT8 DynamicOrderedList [ISCSI_MAX_ATTEMPTS_NUM]
UINT8 DeleteAttemptList [ISCSI_MAX_ATTEMPTS_NUM]
CHAR16 IsId [ISID_CONFIGURABLE_STORAGE]

Field Documentation

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::AttemptName[ATTEMPT_NAME_MAX_SIZE]

UINT8 _ISCSI_CONFIG_IFR_NVDATA::AuthenticationType

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::BootLun[ISCSI_LUN_STR_MAX_LEN]

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::CHAPName[ISCSI_CHAP_NAME_STORAGE]

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::CHAPSecret[ISCSI_CHAP_SECRET_STORAGE]

UINT8 _ISCSI_CONFIG_IFR_NVDATA::CHAPType

UINT8 _ISCSI_CONFIG_IFR_NVDATA::ConnectRetryCount

Referenced by IScsiConvertIfrNvDataToAttemptConfigData().

UINT16 _ISCSI_CONFIG_IFR_NVDATA::ConnectTimeout

UINT8 _ISCSI_CONFIG_IFR_NVDATA::DeleteAttemptList[ISCSI_MAX_ATTEMPTS_NUM]

Referenced by IScsiFormCallback().

UINT8 _ISCSI_CONFIG_IFR_NVDATA::DynamicOrderedList[ISCSI_MAX_ATTEMPTS_NUM]

Referenced by IScsiFormCallback().

UINT8 _ISCSI_CONFIG_IFR_NVDATA::Enabled

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::Gateway[IP4_STR_MAX_SIZE]

UINT8 _ISCSI_CONFIG_IFR_NVDATA::InitiatorInfoFromDhcp

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::InitiatorName[ISCSI_NAME_MAX_SIZE]

Referenced by IScsiFormExtractConfig().

UINT8 _ISCSI_CONFIG_IFR_NVDATA::IpMode

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::IsId[ISID_CONFIGURABLE_STORAGE]

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::KerberosKDCIp[IP_STR_MAX_SIZE]

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::KerberosKDCName[KERBEROS_NAME_MAX_LEN]

UINT16 _ISCSI_CONFIG_IFR_NVDATA::KerberosKDCPort

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::KerberosKDCRealm[KERBEROS_NAME_MAX_LEN]

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::KerberosUserName[KERBEROS_NAME_MAX_LEN]

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::KerberosUserSecret[KERBEROS_SECRET_STORAGE]

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::LocalIp[IP4_STR_MAX_SIZE]

BOOLEAN _ISCSI_CONFIG_IFR_NVDATA::MutualRequired

UINT8 _ISCSI_CONFIG_IFR_NVDATA::Padding1

UINT8 _ISCSI_CONFIG_IFR_NVDATA::Padding2

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::ReverseCHAPName[ISCSI_CHAP_NAME_STORAGE]

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::ReverseCHAPSecret[ISCSI_CHAP_SECRET_STORAGE]

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::SubnetMask[IP4_STR_MAX_SIZE]

UINT8 _ISCSI_CONFIG_IFR_NVDATA::TargetInfoFromDhcp

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::TargetIp[IP_STR_MAX_SIZE]

CHAR16 _ISCSI_CONFIG_IFR_NVDATA::TargetName[ISCSI_NAME_MAX_SIZE]

UINT16 _ISCSI_CONFIG_IFR_NVDATA::TargetPort

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_CONNECTION Struct Reference

Data Fields

UINT32	Signature
LIST_ENTRY	Link
EFI_EVENT	TimeoutEvent
ISCSI_SESSION *	Session
UINT8	State
UINT8	CurrentStage
UINT8	NextStage
UINT8	AuthStep
BOOLEAN	PartialReqSent
BOOLEAN	PartialRspRcvd
BOOLEAN	TransitInitiated
BOOLEAN	ParamNegotiated
UINT16	Cid
UINT32	ExpStatSN
NET_BUF_QUEUE	RspQue
BOOLEAN	Ipv6Flag
TCP_IO	TcpIo
UINT32	MaxRecvDataSegmentLength
ISCSI_DIGEST_TYPE	HeaderDigest
ISCSI_DIGEST_TYPE	DataDigest

Field Documentation

UINT8 _ISCSI_CONNECTION::AuthStep

Referenced by IScsiCreateConnection().

UINT16 _ISCSI_CONNECTION::Cid

Referenced by IScsiCreateConnection().

UINT8 _ISCSI_CONNECTION::CurrentStage

Referenced by IScsiCreateConnection().

ISCSI_DIGEST_TYPE _ISCSI_CONNECTION::DataDigest

Referenced by IScsiCreateConnection().

UINT32 _ISCSI_CONNECTION::ExpStatSN

Referenced by IScsiCreateConnection().

ISCSI_DIGEST_TYPE _ISCSI_CONNECTION::HeaderDigest

Referenced by IScsiCreateConnection().

BOOLEAN _ISCSI_CONNECTION::Ipv6Flag

Referenced by IScsiCreateConnection(), IScsiGetTcpConnDevicePath(), IScsiSessionAbort(), and IScsiSessionLogin().

LIST_ENTRY _ISCSI_CONNECTION::Link

UINT32 _ISCSI_CONNECTION::MaxRecvDataSegmentLength

Referenced by IScsiCreateConnection(), and IScsiGenerateDataOutPduSequence().

UINT8 _ISCSI_CONNECTION::NextStage

Referenced by IScsiCreateConnection().

BOOLEAN _ISCSI_CONNECTION::ParamNegotiated

Referenced by IScsiCreateConnection().

BOOLEAN _ISCSI_CONNECTION::PartialReqSent

Referenced by IScsiCreateConnection().

BOOLEAN _ISCSI_CONNECTION::PartialRspRcvd

Referenced by IScsiCreateConnection().

NET_BUF_QUEUE _ISCSI_CONNECTION::RspQue

Referenced by IScsiCreateConnection().

ISCSI_SESSION* _ISCSI_CONNECTION::Session

UINT32 _ISCSI_CONNECTION::Signature

Referenced by IScsiCreateConnection().

UINT8 _ISCSI_CONNECTION::State

Referenced by IScsiCreateConnection().

TCP_IO _ISCSI_CONNECTION::TcpIo

Referenced by IScsiCreateConnection(), IScsiExecuteScsiCommand(), IScsiGetTcpConnDevicePath(), IScsiSessionAbort(), IScsiSessionLogin(), and IScsiStop().

EFI_EVENT _ISCSI_CONNECTION::TimeoutEvent

Referenced by IScsiCreateConnection(), and IScsiExecuteScsiCommand().

BOOLEAN _ISCSI_CONNECTION::TransitInitiated

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_ISCSI_DRIVER_DATA Struct Reference

Data Fields

UINT32	Signature
EFI_HANDLE	Image
EFI_HANDLE	Controller
ISCSI_PRIVATE_PROTOCOL	IScsiIdentifier
EFI_EVENT	ExitBootServiceEvent
EFI_EXT_SCSI_PASS_THRU_PROTOCOL	IScsiExtScsiPassThru
EFI_EXT_SCSI_PASS_THRU_MODE	ExtScsiPassThruMode
EFI_HANDLE	ExtScsiPassThruHandle
EFI_DEVICE_PATH_PROTOCOL *	DevicePath
EFI_HANDLE	ChildHandle
ISCSI_SESSION *	Session

Field Documentation

EFI_HANDLE _ISCSI_DRIVER_DATA::ChildHandle

Referenced by IScsiStart(), and IScsiStop().

EFI_HANDLE _ISCSI_DRIVER_DATA::Controller

Referenced by IScsiCreateConnection(), IScsiCreateDriverData(), and IScsiStart().

EFI_DEVICE_PATH_PROTOCOL* _ISCSI_DRIVER_DATA::DevicePath

Referenced by IScsiStart().

EFI_EVENT _ISCSI_DRIVER_DATA::ExitBootServiceEvent

Referenced by IScsiCreateDriverData(), and IScsiOnExitBootService().

EFI_HANDLE _ISCSI_DRIVER_DATA::ExtScsiPassThruHandle

Referenced by IScsiCreateDriverData(), IScsiStart(), and IScsiStop().

EFI_EXT_SCSI_PASS_THRU_MODE _ISCSI_DRIVER_DATA::ExtScsiPassThruMode

Referenced by IScsiCreateDriverData().

EFI_HANDLE _ISCSI_DRIVER_DATA::Image

Referenced by IScsiCreateConnection(), IScsiCreateDriverData(), and IScsiStop().

EFI_EXT_SCSI_PASS_THRU_PROTOCOL _ISCSI_DRIVER_DATA::IScsiExtScsiPassThru

Referenced by IScsiCreateDriverData(), and IScsiStart().

ISCSI_PRIVATE_PROTOCOL _ISCSI_DRIVER_DATA::IScsiIdentifier

Referenced by IScsiStart(), and IScsiStop().

ISCSI_SESSION* _ISCSI_DRIVER_DATA::Session

Referenced by IScsiCreateDriverData(), IScsiExecuteScsiCommand(), IScsiExtScsiPassThruBuildDevicePath(), IScsiExtScsiPassThruFunction(), IScsiExtScsiPassThruGetNextTargetLun(), IScsiExtScsiPassThruGetTargetLun(), IScsiOnExitBootService(), IScsiStart(), IScsiStop(), and UpdateName().

UINT32 _ISCSI_DRIVER_DATA::Signature

Referenced by IScsiCreateDriverData().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_FORM_CALLBACK_INFO Struct Reference

Data Fields

UINT32	Signature
EFI_HANDLE	DriverHandle
EFI_HII_CONFIG_ACCESS_PROTOCOL	ConfigAccess
UINT16 *	KeyList
VOID *	FormBuffer
EFI_HII_HANDLE	RegisteredHandle
ISCSI_ATTEMPT_CONFIG_NVDATA *	Current

Field Documentation

EFI_HII_CONFIG_ACCESS_PROTOCOL _ISCSI_FORM_CALLBACK_INFO::ConfigAccess

Referenced by IScsiConfigFormInit(), and IScsiConfigFormUnload().

ISCSI_ATTEMPT_CONFIG_NVDATA* _ISCSI_FORM_CALLBACK_INFO::Current

Referenced by IScsiConfigFormInit(), IScsiConfigProcessDefault(), IScsiFormCallback(), and IScsiFormExtractConfig().

EFI_HANDLE _ISCSI_FORM_CALLBACK_INFO::DriverHandle

Referenced by IScsiConfigFormInit(), IScsiConfigFormUnload(), and IScsiFormExtractConfig().

VOID* _ISCSI_FORM_CALLBACK_INFO::FormBuffer

UINT16* _ISCSI_FORM_CALLBACK_INFO::KeyList

EFI_HII_HANDLE _ISCSI_FORM_CALLBACK_INFO::RegisteredHandle

Referenced by IScsiConfigAddAttempt(), IScsiConfigDisplayDeleteAttempts(), IScsiConfigDisplayOrderAttempts(), IScsiConfigFormInit(), IScsiConfigFormUnload(), IScsiConfigProcessDefault(), IScsiConfigUpdateAttempt(), IScsiConvertIfrNvDataToAttemptConfigData(), and IScsiGetConfigData().

UINT32 _ISCSI_FORM_CALLBACK_INFO::Signature

Referenced by IScsiConfigFormInit().

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_IN_BUFFER_CONTEXT Struct Reference

Data Fields

UINT8 * InData
UINT32 InDataLen

Field Documentation

UINT8* _ISCSI_IN_BUFFER_CONTEXT::InData

Referenced by IScsiExecuteScsiCommand().

UINT32 _ISCSI_IN_BUFFER_CONTEXT::InDataLen

Referenced by IScsiExecuteScsiCommand().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_ISCSI_KEY_VALUE_PAIR Struct Reference

Data Fields

LIST_ENTRY List
    CHAR8 * Key
    CHAR8 * Value

Field Documentation

CHAR8* _ISCSI_KEY_VALUE_PAIR::Key

Referenced by IScsiBuildKeyValueList(), and IScsiGetValueByKeyFromList().

LIST_ENTRY _ISCSI_KEY_VALUE_PAIR::List

Referenced by IScsiBuildKeyValueList(), and IScsiGetValueByKeyFromList().

CHAR8* _ISCSI_KEY_VALUE_PAIR::Value

Referenced by IScsiBuildKeyValueList(), and IScsiGetValueByKeyFromList().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_LOGIN_REQUEST Struct Reference

Data Fields

UINT8 OpCode
UINT8 Flags
UINT8 VersionMax
UINT8 VersionMin
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT8 Isid [6]
UINT16 Tsih
UINT32 InitiatorTaskTag
UINT16 Cid
UINT16 Reserved1
UINT32 CmdSN
UINT32 ExpStatSN
UINT32 Reserved2 [4]

Detailed Description

Login Request.

Field Documentation

UINT16 _ISCSI_LOGIN_REQUEST::Cid

Referenced by IScsiPrepareLoginReq().

UINT32 _ISCSI_LOGIN_REQUEST::CmdSN

Referenced by IScsiPrepareLoginReq().

UINT8 _ISCSI_LOGIN_REQUEST::DataSegmentLength[3]

Referenced by IScsiAddKeyValuePair().

UINT32 _ISCSI_LOGIN_REQUEST::ExpStatSN

Referenced by IScsiPrepareLoginReq().

UINT8 _ISCSI_LOGIN_REQUEST::Flags

UINT32 _ISCSI_LOGIN_REQUEST::InitiatorTaskTag

Referenced by IScsiPrepareLoginReq().

UINT8 _ISCSI_LOGIN_REQUEST::Isid[6]

Referenced by IScsiPrepareLoginReq().

UINT8 _ISCSI_LOGIN_REQUEST::OpCode

UINT16 _ISCSI_LOGIN_REQUEST::Reserved1

UINT32 _ISCSI_LOGIN_REQUEST::Reserved2[4]

UINT8 _ISCSI_LOGIN_REQUEST::TotalAHSLength

UINT16 _ISCSI_LOGIN_REQUEST::Tsih

Referenced by IScsiPrepareLoginReq().

UINT8 _ISCSI_LOGIN_REQUEST::VersionMax

Referenced by IScsiPrepareLoginReq().

UINT8 _ISCSI_LOGIN_REQUEST::VersionMin

Referenced by IScsiPrepareLoginReq().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_LOGIN_RESPONSE Struct Reference

Data Fields

UINT8 OpCode
UINT8 Flags
UINT8 VersionMax
UINT8 VersionActive
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT8 Isid [6]
UINT16 Tsih
UINT32 InitiatorTaskTag
UINT32 Reserved1
UINT32 StatSN
UINT32 ExpCmdSN
UINT32 MaxCmdSN
UINT8 StatusClass
UINT8 StatusDetail
UINT8 Reserved2 [10]

Detailed Description

Login Response.

Field Documentation

UINT8 _ISCSI_LOGIN_RESPONSE::DataSegmentLength[3]

UINT32 _ISCSI_LOGIN_RESPONSE::ExpCmdSN

Referenced by IScsiProcessLoginRsp().

UINT8 _ISCSI_LOGIN_RESPONSE::Flags

UINT32 _ISCSI_LOGIN_RESPONSE::InitiatorTaskTag

Referenced by IScsiProcessLoginRsp().

UINT8 _ISCSI_LOGIN_RESPONSE::Isid[6]

Referenced by IScsiProcessLoginRsp().

UINT32 _ISCSI_LOGIN_RESPONSE::MaxCmdSN

Referenced by IScsiProcessLoginRsp().

UINT8 _ISCSI_LOGIN_RESPONSE::OpCode

UINT32 _ISCSI_LOGIN_RESPONSE::Reserved1

UINT8 _ISCSI_LOGIN_RESPONSE::Reserved2[10]

UINT32 _ISCSI_LOGIN_RESPONSE::StatSN

Referenced by IScsiProcessLoginRsp().

UINT8 _ISCSI_LOGIN_RESPONSE::StatusClass

Referenced by IScsiProcessLoginRsp().

UINT8 _ISCSI_LOGIN_RESPONSE::StatusDetail

UINT8 _ISCSI_LOGIN_RESPONSE::TotalAHSLength

UINT16 _ISCSI_LOGIN_RESPONSE::Tsih

Referenced by IScsiProcessLoginRsp().

UINT8 _ISCSI_LOGIN_RESPONSE::VersionActive

UINT8 _ISCSI_LOGIN_RESPONSE::VersionMax

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_LOGOUT_REQUEST Struct Reference

Data Fields

UINT8 OpCode
UINT8 ReasonCode
UINT16 Reserved1
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT32 Reserved2 [2]
UINT32 InitiatorTaskTag
UINT16 Cid
UINT16 Reserved3
UINT32 CmdSN
UINT32 ExpStatSN
UINT32 Reserved4 [4]

Detailed Description

Logout Request.

Field Documentation

UINT16 _ISCSI_LOGOUT_REQUEST::Cid

UINT32 _ISCSI_LOGOUT_REQUEST::CmdSN

UINT8 _ISCSI_LOGOUT_REQUEST::DataSegmentLength[3]

UINT32 _ISCSI_LOGOUT_REQUEST::ExpStatSN

UINT32 _ISCSI_LOGOUT_REQUEST::InitiatorTaskTag

UINT8 _ISCSI_LOGOUT_REQUEST::OpCode

UINT8 _ISCSI_LOGOUT_REQUEST::ReasonCode

UINT16 _ISCSI_LOGOUT_REQUEST::Reserved1

UINT32 _ISCSI_LOGOUT_REQUEST::Reserved2[2]

UINT16 _ISCSI_LOGOUT_REQUEST::Reserved3

UINT32 _ISCSI_LOGOUT_REQUEST::Reserved4[4]

UINT8 _ISCSI_LOGOUT_REQUEST::TotalAHSLength

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_ISCSI_LOGOUT_RESPONSE Struct Reference

Data Fields

UINT8 OpCode
UINT8 Reserved1
UINT8 Response
UINT8 Reserved2
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT32 Reserved3 [2]
UINT32 InitiatorTaskTag
UINT32 Reserved4
UINT32 StatSN
UINT32 ExpCmdSN
UINT32 MaxCmdSN
UINT32 Reserved5
UINT16 Time2Wait
UINT16 Time2Retain
UINT32 Reserved6

Detailed Description

Logout Response.

Field Documentation

UINT8 _ISCSI_LOGOUT_RESPONSE::DataSegmentLength[3]

UINT32 _ISCSI_LOGOUT_RESPONSE::ExpCmdSN

UINT32 _ISCSI_LOGOUT_RESPONSE::InitiatorTaskTag

UINT32 _ISCSI_LOGOUT_RESPONSE::MaxCmdSN

UINT8 _ISCSI_LOGOUT_RESPONSE::OpCode

UINT8 _ISCSI_LOGOUT_RESPONSE::Reserved1

UINT8 _ISCSI_LOGOUT_RESPONSE::Reserved2

UINT32 _ISCSI_LOGOUT_RESPONSE::Reserved3[2]

UINT32 _ISCSI_LOGOUT_RESPONSE::Reserved4

UINT32 _ISCSI_LOGOUT_RESPONSE::Reserved5

UINT32 _ISCSI_LOGOUT_RESPONSE::Reserved6

UINT8 _ISCSI_LOGOUT_RESPONSE::Response

UINT32 _ISCSI_LOGOUT_RESPONSE::StatSN

UINT16 _ISCSI_LOGOUT_RESPONSE::Time2Retain

UINT16 _ISCSI_LOGOUT_RESPONSE::Time2Wait

UINT8 _ISCSI_LOGOUT_RESPONSE::TotalAHSLength

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

_ISCSI_NOP_IN Struct Reference

Data Fields

UINT8 OpCode
UINT8 Reserved1 [3]
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT8 Lun [8]
UINT32 InitiatorTaskTag
UINT32 TargetTransferTag
UINT32 StatSN
UINT32 ExpCmdSN
UINT32 MaxCmdSN
UINT32 Reserved2 [3]

Detailed Description

NOP-In.

Field Documentation

UINT8 _ISCSI_NOP_IN::DataSegmentLength[3]

UINT32 _ISCSI_NOP_IN::ExpCmdSN

Referenced by IScsiOnNopInRcvd().

UINT32 _ISCSI_NOP_IN::InitiatorTaskTag

Referenced by IScsiOnNopInRcvd().

UINT8 _ISCSI_NOP_IN::Lun[8]

UINT32 _ISCSI_NOP_IN::MaxCmdSN

Referenced by IScsiOnNopInRcvd().

UINT8 _ISCSI_NOP_IN::OpCode

UINT8 _ISCSI_NOP_IN::Reserved1[3]

UINT32 _ISCSI_NOP_IN::Reserved2[3]

UINT32 _ISCSI_NOP_IN::StatSN

Referenced by IScsiOnNopInRcvd().

UINT32 _ISCSI_NOP_IN::TargetTransferTag

UINT8 _ISCSI_NOP_IN::TotalAHSLength

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_NOP_OUT Struct Reference

Data Fields

UINT8 OpCode
UINT8 Reserved1 [3]
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT8 Lun [8]
UINT32 InitiatorTaskTag
UINT32 TargetTransferTag
UINT32 CmdSN
UINT32 ExpStatSN
UINT32 Reserved2 [4]

Detailed Description

NOP-Out.

Field Documentation

UINT32 _ISCSI_NOP_OUT::CmdSN

UINT8 _ISCSI_NOP_OUT::DataSegmentLength[3]

UINT32 _ISCSI_NOP_OUT::ExpStatSN

UINT32 _ISCSI_NOP_OUT::InitiatorTaskTag

UINT8 _ISCSI_NOP_OUT::Lun[8]

UINT8 _ISCSI_NOP_OUT::OpCode

UINT8 _ISCSI_NOP_OUT::Reserved1[3]

UINT32 _ISCSI_NOP_OUT::Reserved2[4]

UINT32 _ISCSI_NOP_OUT::TargetTransferTag

UINT8 _ISCSI_NOP_OUT::TotalAHSLength

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_PRIVATE_PROTOCOL Struct Reference

Data Fields

UINT32 Reserved

Field Documentation

UINT32 _ISCSI_PRIVATE_PROTOCOL::Reserved

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_READY_TO_TRANSFER Struct Reference

Data Fields

UINT8 OpCode
UINT8 Reserved1 [3]
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT8 Lun [8]
UINT32 InitiatorTaskTag
UINT32 TargetTransferTag
UINT32 StatSN
UINT32 ExpCmdSN
UINT32 MaxCmdSN
UINT32 R2TSeqNum
UINT32 BufferOffset
UINT32 DesiredDataTransferLength

Detailed Description

Ready To Transfer.

Field Documentation

UINT32 _ISCSI_READY_TO_TRANSFER::BufferOffset

Referenced by IScsiOnR2TRcvd().

UINT8 _ISCSI_READY_TO_TRANSFER::DataSegmentLength[3]

UINT32 _ISCSI_READY_TO_TRANSFER::DesiredDataTransferLength

Referenced by IScsiOnR2TRcvd().

UINT32 _ISCSI_READY_TO_TRANSFER::ExpCmdSN

UINT32 _ISCSI_READY_TO_TRANSFER::InitiatorTaskTag

Referenced by IScsiOnR2TRcvd().

UINT8 _ISCSI_READY_TO_TRANSFER::Lun[8]

UINT32 _ISCSI_READY_TO_TRANSFER::MaxCmdSN

UINT8 _ISCSI_READY_TO_TRANSFER::OpCode

UINT32 _ISCSI_READY_TO_TRANSFER::R2TSeqNum

Referenced by IScsiOnR2TRcvd().

UINT8 _ISCSI_READY_TO_TRANSFER::Reserved1[3]

UINT32 _ISCSI_READY_TO_TRANSFER::StatSN

Referenced by IScsiOnR2TRcvd().

UINT32 _ISCSI_READY_TO_TRANSFER::TargetTransferTag

Referenced by IScsiOnR2TRcvd().

UINT8 _ISCSI_READY_TO_TRANSFER::TotalAHSLength

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_REJECT Struct Reference

Data Fields

UINT8 OpCode
UINT8 Reserved1
UINT8 Reason
UINT8 Reserved2
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT32 Reserved3 [2]
UINT32 InitiatorTaskTag
UINT32 Reserved4
UINT32 StatSN
UINT32 ExpCmdSN
UINT32 MaxCmdSN
UINT32 DataSN
UINT32 Reserved5 [2]

Detailed Description

Reject.

Field Documentation

UINT8 _ISCSI_REJECT::DataSegmentLength[3]

UINT32 _ISCSI_REJECT::DataSN

UINT32 _ISCSI_REJECT::ExpCmdSN

UINT32 _ISCSI_REJECT::InitiatorTaskTag

UINT32 _ISCSI_REJECT::MaxCmdSN

UINT8 _ISCSI_REJECT::OpCode

UINT8 _ISCSI_REJECT::Reason

UINT8 _ISCSI_REJECT::Reserved1

UINT8 _ISCSI_REJECT::Reserved2

UINT32 _ISCSI_REJECT::Reserved3[2]

UINT32 _ISCSI_REJECT::Reserved4

UINT32 _ISCSI_REJECT::Reserved5[2]

UINT32 _ISCSI_REJECT::StatSN

UINT8 _ISCSI_REJECT::TotalAHSLength

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_ROOT_PATH_FIELD Struct Reference

Data Fields

CHAR8 * Str
UINT8 Len

Field Documentation

UINT8 _ISCSI_ROOT_PATH_FIELD::Len

Referenced by IScsiDhcp6ExtractRootPath(), and IScsiDhcpExtractRootPath().

CHAR8* _ISCSI_ROOT_PATH_FIELD::Str

Referenced by IScsiDhcp6ExtractRootPath(), and IScsiDhcpExtractRootPath().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

_ISCSI_SCSI_DATA_IN Struct Reference

Data Fields

UINT8 OpCode
UINT8 Flags
UINT8 Reserved1
UINT8 Status
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT8 Lun [8]
UINT32 InitiatorTaskTag
UINT32 TargetTransferTag
UINT32 StatSN
UINT32 ExpCmdSN
UINT32 MaxCmdSN
UINT32 DataSN
UINT32 BufferOffset
UINT32 ResidualCount

Detailed Description

SCSI Data-In

Field Documentation

UINT32 _ISCSI_SCSI_DATA_IN::BufferOffset

UINT8 _ISCSI_SCSI_DATA_IN::DataSegmentLength[3]

UINT32 _ISCSI_SCSI_DATA_IN::DataSN

Referenced by IScsiOnDataInRcvd().

UINT32 _ISCSI_SCSI_DATA_IN::ExpCmdSN

Referenced by IScsiOnDataInRcvd().

UINT8 _ISCSI_SCSI_DATA_IN::Flags

UINT32 _ISCSI_SCSI_DATA_IN::InitiatorTaskTag

Referenced by IScsiOnDataInRcvd().

UINT8 _ISCSI_SCSI_DATA_IN::Lun[8]

UINT32 _ISCSI_SCSI_DATA_IN::MaxCmdSN

Referenced by IScsiOnDataInRcvd().

UINT8 _ISCSI_SCSI_DATA_IN::OpCode

UINT8 _ISCSI_SCSI_DATA_IN::Reserved1

UINT32 _ISCSI_SCSI_DATA_IN::ResidualCount

Referenced by IScsiOnDataInRcvd().

UINT32 _ISCSI_SCSI_DATA_IN::StatSN

Referenced by IScsiOnDataInRcvd().

UINT8 _ISCSI_SCSI_DATA_IN::Status

Referenced by IScsiOnDataInRcvd().

UINT32 _ISCSI_SCSI_DATA_IN::TargetTransferTag

UINT8 _ISCSI_SCSI_DATA_IN::TotalAHSLength

_ISCSI_SCSI_DATA_OUT Struct Reference

Data Fields

UINT8 OpCode
UINT8 Reserved1 [3]
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT8 Lun [8]
UINT32 InitiatorTaskTag
UINT32 TargetTransferTag
UINT32 Reserved2
UINT32 ExpStatSN
UINT32 Reserved3
UINT32 DataSN
UINT32 BufferOffset
UINT32 Reserved4

Detailed Description

SCSI Data-Out

Field Documentation

UINT32 _ISCSI_SCSI_DATA_OUT::BufferOffset

Referenced by IScsiNewDataOutPdu().

UINT8 _ISCSI_SCSI_DATA_OUT::DataSegmentLength[3]

UINT32 _ISCSI_SCSI_DATA_OUT::DataSN

Referenced by IScsiNewDataOutPdu().

UINT32 _ISCSI_SCSI_DATA_OUT::ExpStatSN

Referenced by IScsiNewDataOutPdu().

UINT32 _ISCSI_SCSI_DATA_OUT::InitiatorTaskTag

Referenced by IScsiNewDataOutPdu().

UINT8 _ISCSI_SCSI_DATA_OUT::Lun[8]

Referenced by IScsiNewDataOutPdu().

UINT8 _ISCSI_SCSI_DATA_OUT::OpCode

UINT8 _ISCSI_SCSI_DATA_OUT::Reserved1[3]

UINT32 _ISCSI_SCSI_DATA_OUT::Reserved2

UINT32 _ISCSI_SCSI_DATA_OUT::Reserved3

UINT32 _ISCSI_SCSI_DATA_OUT::Reserved4

UINT32 _ISCSI_SCSI_DATA_OUT::TargetTransferTag

Referenced by IScsiNewDataOutPdu().

UINT8 _ISCSI_SCSI_DATA_OUT::TotalAHSLength

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_SENSE_DATA Struct Reference

Data Fields

UINT16 Length
UINT8 Data [2]

Field Documentation

UINT8 _ISCSI_SENSE_DATA::Data[2]

Referenced by IScsiOnScsiRspRcvd().

UINT16 _ISCSI_SENSE_DATA::Length

Referenced by IScsiOnScsiRspRcvd().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

_ISCSI_SESSION Struct Reference

Data Fields

```
                                  UINT32 Signature
                   ISCSI_DRIVER_DATA * Private
ISCSI_ATTEMPT_CONFIG_NVDATA * ConfigData
                                   UINT8 AuthType
union {
  ISCSI_CHAP_AUTH_DATA   CHAP
}                                        AuthData
                                   UINT8 State
                                   UINT8 Isid [6]
                                  UINT16 Tsih
                                  UINT32 CmdSN
                                  UINT32 ExpCmdSN
                                  UINT32 MaxCmdSN
                                  UINT32 InitiatorTaskTag
                                  UINT16 NextCid
                             LIST_ENTRY Conns
                                  UINT32 NumConns
                             LIST_ENTRY TcbList
                                  UINT16 TargetPortalGroupTag
                                  UINT32 MaxConnections
                               BOOLEAN InitialR2T
                               BOOLEAN ImmediateData
                                  UINT32 MaxBurstLength
                                  UINT32 FirstBurstLength
                                  UINT32 DefaultTime2Wait
                                  UINT32 DefaultTime2Retain
                                  UINT16 MaxOutstandingR2T
                               BOOLEAN DataPDUInOrder
                               BOOLEAN DataSequenceInOrder
                                   UINT8 ErrorRecoveryLevel
```

Field Documentation

union { ... } _ISCSI_SESSION::AuthData

Referenced by IScsiCHAPOnRspReceived(), IScsiCHAPToSendReq(), IScsiExtScsiPassThruBuildDevicePath(), and IScsiStart().

UINT8 _ISCSI_SESSION::AuthType

Referenced by IScsiCHAPOnRspReceived(), IScsiCHAPToSendReq(), IScsiExtScsiPassThruBuildDevicePath(), IScsiPrepareLoginReq(), IScsiProcessLoginRsp(), and IScsiStart().

ISCSI_CHAP_AUTH_DATA _ISCSI_SESSION::CHAP

Referenced by IScsiCHAPOnRspReceived(), IScsiCHAPToSendReq(), IScsiExtScsiPassThruBuildDevicePath(), and IScsiStart().

UINT32 _ISCSI_SESSION::CmdSN

Referenced by IScsiNewTcb(), IScsiPrepareLoginReq(), and IScsiProcessLoginRsp().

ISCSI_ATTEMPT_CONFIG_NVDATA* _ISCSI_SESSION::ConfigData

Referenced by IScsiCHAPToSendReq(), IScsiExtScsiPassThruBuildDevicePath(), IScsiExtScsiPassThruGetNextTargetLun(), IScsiExtScsiPassThruGetTargetLun(), IScsiStart(), and UpdateName().

LIST_ENTRY _ISCSI_SESSION::Conns

Referenced by IScsiExecuteScsiCommand(), and IScsiStop().

BOOLEAN _ISCSI_SESSION::DataPDUInOrder

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

BOOLEAN _ISCSI_SESSION::DataSequenceInOrder

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

UINT32 _ISCSI_SESSION::DefaultTime2Retain

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

UINT32 _ISCSI_SESSION::DefaultTime2Wait

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

UINT8 _ISCSI_SESSION::ErrorRecoveryLevel

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

UINT32 _ISCSI_SESSION::ExpCmdSN

Referenced by IScsiProcessLoginRsp().

UINT32 _ISCSI_SESSION::FirstBurstLength

Referenced by IScsiCheckOpParams(), IScsiExecuteScsiCommand(), IScsiFillOpParams(), and IScsiNewScsiCmdPdu().

BOOLEAN _ISCSI_SESSION::ImmediateData

Referenced by IScsiCheckOpParams(), IScsiFillOpParams(), and IScsiNewScsiCmdPdu().

BOOLEAN _ISCSI_SESSION::InitialR2T

Referenced by IScsiCheckOpParams(), IScsiExecuteScsiCommand(), IScsiFillOpParams(), and IScsiNewScsiCmdPdu().

UINT32 _ISCSI_SESSION::InitiatorTaskTag

Referenced by IScsiNewTcb(), IScsiPrepareLoginReq(), and IScsiProcessLoginRsp().

UINT8 _ISCSI_SESSION::Isid[6]

Referenced by IScsiPrepareLoginReq(), and IScsiProcessLoginRsp().

UINT32 _ISCSI_SESSION::MaxBurstLength

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

UINT32 _ISCSI_SESSION::MaxCmdSN

Referenced by IScsiNewTcb(), and IScsiProcessLoginRsp().

UINT32 _ISCSI_SESSION::MaxConnections

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

UINT16 _ISCSI_SESSION::MaxOutstandingR2T

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

UINT16 _ISCSI_SESSION::NextCid

UINT32 _ISCSI_SESSION::NumConns

ISCSI_DRIVER_DATA* _ISCSI_SESSION::Private

Referenced by IScsiStart().

UINT32 _ISCSI_SESSION::Signature

UINT8 _ISCSI_SESSION::State

Referenced by IScsiExecuteScsiCommand(), and IScsiProcessLoginRsp().

UINT16 _ISCSI_SESSION::TargetPortalGroupTag

Referenced by IScsiCHAPOnRspReceived(), and IScsiExtScsiPassThruBuildDevicePath().

LIST_ENTRY _ISCSI_SESSION::TcbList

Referenced by IScsiNewTcb().

UINT16 _ISCSI_SESSION::Tsih

Referenced by IScsiPrepareLoginReq(), and IScsiProcessLoginRsp().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_SESSION_CONFIG_NVDATA Struct Reference

Data Fields

UINT16	TargetPort
UINT8	Enabled
UINT8	IpMode
EFI_IP_ADDRESS	LocalIp
EFI_IPv4_ADDRESS	SubnetMask
EFI_IP_ADDRESS	Gateway
BOOLEAN	InitiatorInfoFromDhcp
BOOLEAN	TargetInfoFromDhcp
CHAR8	TargetName [ISCSI_NAME_MAX_SIZE]
EFI_IP_ADDRESS	TargetIp
UINT8	PrefixLength
UINT8	BootLun [8]
UINT16	ConnectTimeout timout value in milliseconds
UINT8	ConnectRetryCount
UINT8	IsId [6]

Field Documentation

UINT8 _ISCSI_SESSION_CONFIG_NVDATA::BootLun[8]

Referenced by IScsiConvertAttemptConfigDataToIfrNvData(), IScsiDhcp6ExtractRootPath(), IScsiDhcpExtractRootPath(), IScsiExtScsiPassThruBuildDevicePath(), IScsiExtScsiPassThruGetNextTargetLun(), IScsiExtScsiPassThruGetTargetLun(), IScsiFillNICAndTargetSections(), and IScsiFormCallback().

UINT8 _ISCSI_SESSION_CONFIG_NVDATA::ConnectRetryCount

Referenced by IScsiConfigProcessDefault(), and IScsiConvertAttemptConfigDataToIfrNvData().

UINT16 _ISCSI_SESSION_CONFIG_NVDATA::ConnectTimeout
timout value in milliseconds

Referenced by IScsiConfigProcessDefault(), and IScsiConvertAttemptConfigDataToIfrNvData().

UINT8 _ISCSI_SESSION_CONFIG_NVDATA::Enabled

Referenced by IScsiConfigDeleteAttempts(), IScsiConfigGetAttemptByNic(), IScsiConvertAttemptConfigDataToIfrNvData(), IScsiFillNICAndTargetSections(), IScsiGetConfigData(), IScsiRemoveNic(), and IScsiStart().

EFI_IP_ADDRESS _ISCSI_SESSION_CONFIG_NVDATA::Gateway

Referenced by IScsiConvertAttemptConfigDataToIfrNvData(), IScsiCreateConnection(), IScsiFillNICAndTargetSections(), IScsiFormCallback(), IScsiGetIp6NicInfo(), and IScsiParseDhcpAck().

BOOLEAN _ISCSI_SESSION_CONFIG_NVDATA::InitiatorInfoFromDhcp

Referenced by IScsiConvertAttemptConfigDataToIfrNvData(), IScsiFillNICAndTargetSections(), IScsiGetConfigData(), and IScsiStart().

UINT8 _ISCSI_SESSION_CONFIG_NVDATA::IpMode

Referenced by IScsiConvertAttemptConfigDataToIfrNvData(), IScsiCreateConnection(), IScsiDhcp6ExtractRootPath(), IScsiDhcpExtractRootPath(), IScsiFillNICAndTargetSections(), IScsiGetConfigData(), and IScsiStart().

UINT8 _ISCSI_SESSION_CONFIG_NVDATA::IsId[6]

Referenced by IScsiConfigProcessDefault(), IScsiConvertAttemptConfigDataToIfrNvData(), and IScsiFormCallback().

EFI_IP_ADDRESS _ISCSI_SESSION_CONFIG_NVDATA::LocalIp

Referenced by IScsiConvertAttemptConfigDataToIfrNvData(), IScsiCreateConnection(), IScsiFillNICAndTargetSections(), IScsiFormCallback(), IScsiGetIp6NicInfo(), and IScsiParseDhcpAck().

UINT8 _ISCSI_SESSION_CONFIG_NVDATA::PrefixLength

Referenced by IScsiFillNICAndTargetSections(), and IScsiGetIp6NicInfo().

EFI_IPv4_ADDRESS _ISCSI_SESSION_CONFIG_NVDATA::SubnetMask

Referenced by IScsiConvertAttemptConfigDataToIfrNvData(), IScsiCreateConnection(), IScsiFillNICAndTargetSections(), IScsiFormCallback(), and IScsiParseDhcpAck().

BOOLEAN _ISCSI_SESSION_CONFIG_NVDATA::TargetInfoFromDhcp

Referenced by IScsiConvertAttemptConfigDataToIfrNvData(), IScsiDoDhcp(), and IScsiGetConfigData().

EFI_IP_ADDRESS _ISCSI_SESSION_CONFIG_NVDATA::TargetIp

Referenced by IScsiConvertAttemptConfigDataToIfrNvData(), IScsiCreateConnection(), IScsiDhcp6ExtractRootPath(), IScsiDhcpExtractRootPath(), IScsiFillNICAndTargetSections(), IScsiFormCallback(), and IScsiGetIp6NicInfo().

CHAR8 _ISCSI_SESSION_CONFIG_NVDATA::TargetName[ISCSI_NAME_MAX_SIZE]

Referenced by IScsiCHAPToSendReq(), IScsiConvertAttemptConfigDataToIfrNvData(), IScsiDhcp6ExtractRootPath(), IScsiDhcpExtractRootPath(), IScsiExtScsiPassThruBuildDevicePath(), IScsiExtScsiPassThruGetTargetLun(), IScsiFillNICAndTargetSections(), and IScsiFormCallback().

UINT16 _ISCSI_SESSION_CONFIG_NVDATA::TargetPort

Referenced by IScsiConfigProcessDefault(), IScsiConvertAttemptConfigDataToIfrNvData(), IScsiCreateConnection(), IScsiDhcp6ExtractRootPath(), IScsiDhcpExtractRootPath(), and IScsiFillNICAndTargetSections().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_SNACK_REQUEST Struct Reference

Data Fields

UINT8 OpCode
UINT8 Type
UINT16 Reserved1
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT8 Lun [8]
UINT32 InitiatorTaskTag
UINT32 TargetTransferTag
UINT32 Reserved2
UINT32 ExpStatSN
UINT32 Reserved [2]
UINT32 BegRun
UINT32 RunLength

Detailed Description

SNACK Request.

Field Documentation

UINT32 _ISCSI_SNACK_REQUEST::BegRun

UINT8 _ISCSI_SNACK_REQUEST::DataSegmentLength[3]

UINT32 _ISCSI_SNACK_REQUEST::ExpStatSN

UINT32 _ISCSI_SNACK_REQUEST::InitiatorTaskTag

UINT8 _ISCSI_SNACK_REQUEST::Lun[8]

UINT8 _ISCSI_SNACK_REQUEST::OpCode

UINT32 _ISCSI_SNACK_REQUEST::Reserved[2]

UINT16 _ISCSI_SNACK_REQUEST::Reserved1

UINT32 _ISCSI_SNACK_REQUEST::Reserved2

UINT32 _ISCSI_SNACK_REQUEST::RunLength

UINT32 _ISCSI_SNACK_REQUEST::TargetTransferTag

UINT8 _ISCSI_SNACK_REQUEST::TotalAHSLength

UINT8 _ISCSI_SNACK_REQUEST::Type

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

_ISCSI_TCB Struct Reference

Data Fields

```
            LIST_ENTRY Link
               BOOLEAN SoFarInOrder
                 UINT32 ExpDataSN
               BOOLEAN FbitReceived
               BOOLEAN StatusXferd
                 UINT32 ActiveR2Ts
                 UINT32 Response
                CHAR8 * Reason
                 UINT32 InitiatorTaskTag
                 UINT32 CmdSN
                 UINT32 SNACKTag
ISCSI_XFER_CONTEXT XferContext
 ISCSI_CONNECTION * Conn
```

Field Documentation

UINT32 _ISCSI_TCB::ActiveR2Ts

UINT32 _ISCSI_TCB::CmdSN

Referenced by IScsiNewTcb().

ISCSI_CONNECTION* _ISCSI_TCB::Conn

Referenced by IScsiNewTcb().

UINT32 _ISCSI_TCB::ExpDataSN

BOOLEAN _ISCSI_TCB::FbitReceived

UINT32 _ISCSI_TCB::InitiatorTaskTag

Referenced by IScsiFindTcbByITT(), and IScsiNewTcb().

LIST_ENTRY _ISCSI_TCB::Link

Referenced by IScsiNewTcb().

CHAR8* _ISCSI_TCB::Reason

UINT32 _ISCSI_TCB::Response

UINT32 _ISCSI_TCB::SNACKTag

BOOLEAN _ISCSI_TCB::SoFarInOrder

Referenced by IScsiNewTcb().

BOOLEAN _ISCSI_TCB::StatusXferd

Referenced by IScsiExecuteScsiCommand().

ISCSI_XFER_CONTEXT _ISCSI_TCB::XferContext

Referenced by IScsiExecuteScsiCommand().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_TMF_REQUEST Struct Reference

Data Fields

UINT8 OpCode
UINT8 Fuction
UINT16 Reserved1
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT8 Lun [8]
UINT32 InitiatorTaskTag
UINT32 ReferencedTaskTag
UINT32 CmdSN
UINT32 ExpStatSN
UINT32 RefCmdSN
UINT32 ExpDataSN
UINT32 Reserved2 [2]

Detailed Description

iSCSI Task Managment Function Request.

Field Documentation

UINT32 _ISCSI_TMF_REQUEST::CmdSN

UINT8 _ISCSI_TMF_REQUEST::DataSegmentLength[3]

UINT32 _ISCSI_TMF_REQUEST::ExpDataSN

UINT32 _ISCSI_TMF_REQUEST::ExpStatSN

UINT8 _ISCSI_TMF_REQUEST::Fuction

UINT32 _ISCSI_TMF_REQUEST::InitiatorTaskTag

UINT8 _ISCSI_TMF_REQUEST::Lun[8]

UINT8 _ISCSI_TMF_REQUEST::OpCode

UINT32 _ISCSI_TMF_REQUEST::RefCmdSN

UINT32 _ISCSI_TMF_REQUEST::ReferencedTaskTag

UINT16 _ISCSI_TMF_REQUEST::Reserved1

UINT32 _ISCSI_TMF_REQUEST::Reserved2[2]

UINT8 _ISCSI_TMF_REQUEST::TotalAHSLength

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

_ISCSI_TMF_RESPONSE Struct Reference

Data Fields

UINT8 OpCode
UINT8 Reserved1
UINT8 Response
UINT8 Reserved2
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT32 Reserver3 [2]
UINT32 InitiatorTaskTag
UINT32 Reserved4
UINT32 StatSN
UINT32 ExpCmdSN
UINT32 MaxCmdSN
UINT32 Reserved [3]

Detailed Description

iSCSI Task Management Function Response.

Field Documentation

UINT8 _ISCSI_TMF_RESPONSE::DataSegmentLength[3]

UINT32 _ISCSI_TMF_RESPONSE::ExpCmdSN

UINT32 _ISCSI_TMF_RESPONSE::InitiatorTaskTag

UINT32 _ISCSI_TMF_RESPONSE::MaxCmdSN

UINT8 _ISCSI_TMF_RESPONSE::OpCode

UINT32 _ISCSI_TMF_RESPONSE::Reserved[3]

UINT8 _ISCSI_TMF_RESPONSE::Reserved1

UINT8 _ISCSI_TMF_RESPONSE::Reserved2

UINT32 _ISCSI_TMF_RESPONSE::Reserved4

UINT32 _ISCSI_TMF_RESPONSE::Reserver3[2]

UINT8 _ISCSI_TMF_RESPONSE::Response

UINT32 _ISCSI_TMF_RESPONSE::StatSN

UINT8 _ISCSI_TMF_RESPONSE::TotalAHSLength

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_ISCSI_XFER_CONTEXT Struct Reference

Data Fields

UINT32 TargetTransferTag
UINT32 Offset
UINT32 DesiredLength
UINT32 ExpDataSN

Field Documentation

UINT32 _ISCSI_XFER_CONTEXT::DesiredLength

Referenced by IScsiExecuteScsiCommand(), IScsiGenerateDataOutPduSequence(), and IScsiOnR2TRcvd().

UINT32 _ISCSI_XFER_CONTEXT::ExpDataSN

UINT32 _ISCSI_XFER_CONTEXT::Offset

Referenced by IScsiExecuteScsiCommand(), IScsiGenerateDataOutPduSequence(), IScsiNewDataOutPdu(), and IScsiOnR2TRcvd().

UINT32 _ISCSI_XFER_CONTEXT::TargetTransferTag

Referenced by IScsiExecuteScsiCommand(), IScsiNewDataOutPdu(), and IScsiOnR2TRcvd().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

_MTFTP6_GETINFO_CONTEXT Struct Reference

Data Fields

EFI_MTFTP6_PACKET ** Packet
UINT32 * PacketLen
EFI_STATUS Status

Field Documentation

EFI_MTFTP6_PACKET** _MTFTP6_GETINFO_CONTEXT::Packet

Referenced by EfiMtftp6GetInfo(), and Mtftp6CheckPacket().

UINT32* _MTFTP6_GETINFO_CONTEXT::PacketLen

Referenced by EfiMtftp6GetInfo(), and Mtftp6CheckPacket().

EFI_STATUS _MTFTP6_GETINFO_CONTEXT::Status

Referenced by EfiMtftp6GetInfo(), and Mtftp6CheckPacket().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_MTFTP6_INSTANCE Struct Reference

Data Fields

UINT32	Signature
EFI_HANDLE	Handle
LIST_ENTRY	Link
EFI_MTFTP6_PROTOCOL	Mtftp6
MTFTP6_SERVICE *	Service
EFI_MTFTP6_CONFIG_DATA *	Config
EFI_MTFTP6_TOKEN *	Token
MTFTP6_EXT_OPTION_INFO	ExtInfo
UINT16	BlkSize
UINT16	LastBlk
LIST_ENTRY	BlkList
EFI_IPv6_ADDRESS	ServerIp
UINT16	ServerCmdPort
UINT16	ServerDataPort
UDP_IO *	UdpIo
EFI_IPv6_ADDRESS	McastIp
UINT16	McastPort
UDP_IO *	McastUdpIo
NET_BUF *	LastPacket
UINT32	CurRetry
UINT32	MaxRetry
UINT32	PacketToLive
UINT32	Timeout
EFI_TPL	OldTpl
BOOLEAN	IsTransmitted
BOOLEAN	IsMaster
BOOLEAN	InDestroy

Field Documentation

LIST_ENTRY _MTFTP6_INSTANCE::BlkList

Referenced by Mtftp6CreateInstance().

UINT16 _MTFTP6_INSTANCE::BlkSize

Referenced by Mtftp6OperationStart(), and Mtftp6RrqInput().

EFI_MTFTP6_CONFIG_DATA* _MTFTP6_INSTANCE::Config

Referenced by EfiMtftp6Configure(), EfiMtftp6GetModeData(), EfiMtftp6Poll(), and Mtftp6OperationStart().

UINT32 _MTFTP6_INSTANCE::CurRetry

Referenced by Mtftp6OnTimerTick().

MTFTP6_EXT_OPTION_INFO _MTFTP6_INSTANCE::ExtInfo

Referenced by Mtftp6OperationStart().

EFI_HANDLE _MTFTP6_INSTANCE::Handle

Referenced by EfiMtftp6Configure(), Mtftp6DestroyChildEntryInHandleBuffer(), Mtftp6ServiceBindingCreateChild(), and Mtftp6ServiceBindingDestroyChild().

BOOLEAN _MTFTP6_INSTANCE::InDestroy

Referenced by Mtftp6CreateInstance(), and Mtftp6ServiceBindingDestroyChild().

BOOLEAN _MTFTP6_INSTANCE::IsMaster

Referenced by Mtftp6OperationStart().

BOOLEAN _MTFTP6_INSTANCE::IsTransmitted

UINT16 _MTFTP6_INSTANCE::LastBlk

NET_BUF* _MTFTP6_INSTANCE::LastPacket

Referenced by Mtftp6OnTimerTick().

LIST_ENTRY _MTFTP6_INSTANCE::Link

Referenced by Mtftp6CreateInstance(), Mtftp6ServiceBindingCreateChild(), and Mtftp6ServiceBindingDestroyChild().

UINT32 _MTFTP6_INSTANCE::MaxRetry

Referenced by Mtftp6OnTimerTick(), and Mtftp6OperationStart().

EFI_IPv6_ADDRESS _MTFTP6_INSTANCE::McastIp

Referenced by Mtftp6RrqConfigMcastUdpIo(), and Mtftp6RrqInput().

UINT16 _MTFTP6_INSTANCE::McastPort

Referenced by Mtftp6RrqConfigMcastUdpIo().

UDP_IO* _MTFTP6_INSTANCE::McastUdpIo

Referenced by Mtftp6RrqInput(), and Mtftp6ServiceBindingDestroyChild().

EFI_MTFTP6_PROTOCOL _MTFTP6_INSTANCE::Mtftp6

Referenced by Mtftp6CreateInstance(), Mtftp6OnTimerTick(), Mtftp6RrqInput(), Mtftp6ServiceBindingCreateChild(), and Mtftp6WrqInput().

EFI_TPL _MTFTP6_INSTANCE::OldTpl

Referenced by Mtftp6OperationStart().

UINT32 _MTFTP6_INSTANCE::PacketToLive

Referenced by Mtftp6OnTimerTick().

UINT16 _MTFTP6_INSTANCE::ServerCmdPort

Referenced by Mtftp6OperationStart().

UINT16 _MTFTP6_INSTANCE::ServerDataPort

Referenced by Mtftp6OperationStart(), Mtftp6RrqInput(), and Mtftp6WrqInput().

EFI_IPv6_ADDRESS _MTFTP6_INSTANCE::ServerIp

Referenced by Mtftp6OperationStart(), and Mtftp6RrqConfigMcastUdpIo().

MTFTP6_SERVICE* _MTFTP6_INSTANCE::Service

Referenced by EfiMtftp6Configure(), Mtftp6CreateInstance(), and Mtftp6ServiceBindingDestroyChild().

UINT32 _MTFTP6_INSTANCE::Signature

Referenced by Mtftp6CreateInstance().

UINT32 _MTFTP6_INSTANCE::Timeout

Referenced by Mtftp6OperationStart().

EFI_MTFTP6_TOKEN* _MTFTP6_INSTANCE::Token

Referenced by Mtftp6OnTimerTick(), Mtftp6OperationStart(), Mtftp6RrqInput(), and Mtftp6WrqInput().

UDP_IO* _MTFTP6_INSTANCE::UdpIo

Referenced by EfiMtftp6Configure(), EfiMtftp6Poll(), Mtftp6RrqInput(), Mtftp6ServiceBindingDestroyChild(), and Mtftp6WrqInput().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_MTFTP6_SERVICE Struct Reference

Data Fields

UINT32	Signature
EFI_SERVICE_BINDING_PROTOCOL	ServiceBinding
EFI_HANDLE	Controller
EFI_HANDLE	Image
UINT16	ChildrenNum
LIST_ENTRY	Children
EFI_EVENT	Timer
UDP_IO *	DummyUdpIo

Field Documentation

LIST_ENTRY _MTFTP6_SERVICE::Children

Referenced by Mtftp6CreateService(), Mtftp6DriverBindingStop(), Mtftp6OnTimerTick(), and Mtftp6ServiceBindingCreateChild().

UINT16 _MTFTP6_SERVICE::ChildrenNum

Referenced by Mtftp6CreateService(), Mtftp6ServiceBindingCreateChild(), and Mtftp6ServiceBindingDestroyChild().

EFI_HANDLE _MTFTP6_SERVICE::Controller

Referenced by EfiMtftp6Configure(), and Mtftp6CreateService().

UDP_IO* _MTFTP6_SERVICE::DummyUdpIo

Referenced by Mtftp6CreateService(), Mtftp6ServiceBindingCreateChild(), and Mtftp6ServiceBindingDestroyChild().

EFI_HANDLE _MTFTP6_SERVICE::Image

Referenced by EfiMtftp6Configure(), and Mtftp6CreateService().

EFI_SERVICE_BINDING_PROTOCOL _MTFTP6_SERVICE::ServiceBinding

Referenced by Mtftp6CreateService(), and Mtftp6DriverBindingStart().

UINT32 _MTFTP6_SERVICE::Signature

Referenced by Mtftp6CreateService().

EFI_EVENT _MTFTP6_SERVICE::Timer

Referenced by Mtftp6CreateService(), and Mtftp6DriverBindingStart().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_NET_PROTOCOL Union Reference

Data Fields

EFI_TCP4_PROTOCOL Tcp4Protocol
Tcp4 protocol.
EFI_TCP6_PROTOCOL Tcp6Protocol
Tcp6 protocol.

Detailed Description

The union type of TCP and UDP protocol.

Field Documentation

EFI_TCP4_PROTOCOL _NET_PROTOCOL::Tcp4Protocol
Tcp4 protocol.

EFI_TCP6_PROTOCOL _NET_PROTOCOL::Tcp6Protocol
Tcp6 protocol.

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

_PING6_ICMP6_TX_INFO Struct Reference

Data Fields

LIST_ENTRY	Link
UINT16	SequenceNum
UINT64	TimeStamp
EFI_IP6_COMPLETION_TOKEN *	Token

Field Documentation

LIST_ENTRY _PING6_ICMP6_TX_INFO::Link

Referenced by Ping6(), Ping6MatchEchoReply(), Ping6OnTimerRoutine(), and Ping6SendEchoRequest().

UINT16 _PING6_ICMP6_TX_INFO::SequenceNum

Referenced by Ping6MatchEchoReply(), Ping6OnTimerRoutine(), and Ping6SendEchoRequest().

UINT64 _PING6_ICMP6_TX_INFO::TimeStamp

Referenced by Ping6MatchEchoReply(), Ping6OnTimerRoutine(), and Ping6SendEchoRequest().

EFI_IP6_COMPLETION_TOKEN* _PING6_ICMP6_TX_INFO::Token

Referenced by Ping6(), Ping6OnTimerRoutine(), and Ping6SendEchoRequest().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

_PING6_PRIVATE_DATA Struct Reference

Data Fields

EFI_HANDLE	ImageHandle
EFI_HANDLE	NicHandle
EFI_HANDLE	Ip6ChildHandle
EFI_IP6_PROTOCOL *	Ip6
EFI_EVENT	Timer
EFI_STATUS	Status
LIST_ENTRY	TxList
EFI_IP6_COMPLETION_TOKEN	RxToken
UINT16	RxCount
UINT16	TxCount
UINT64	RttSum
UINT64	RttMin
UINT64	RttMax
UINT32	SequenceNum
EFI_IPv6_ADDRESS	SrcAddress
EFI_IPv6_ADDRESS	DstAddress
UINT32	SendNum
UINT32	BufferSize

Field Documentation

UINT32 _PING6_PRIVATE_DATA::BufferSize

Referenced by Ping6(), and Ping6OnEchoReplyReceived().

EFI_IPv6_ADDRESS _PING6_PRIVATE_DATA::DstAddress

Referenced by Ping6(), and Ping6OnEchoReplyReceived().

EFI_HANDLE _PING6_PRIVATE_DATA::ImageHandle

Referenced by Ping6().

EFI_IP6_PROTOCOL* _PING6_PRIVATE_DATA::Ip6

Referenced by Ping6(), Ping6OnEchoReplyReceived(), and Ping6OnTimerRoutine().

EFI_HANDLE _PING6_PRIVATE_DATA::Ip6ChildHandle

Referenced by Ping6().

EFI_HANDLE _PING6_PRIVATE_DATA::NicHandle

UINT64 _PING6_PRIVATE_DATA::RttMax

Referenced by Ping6(), and Ping6OnEchoReplyReceived().

UINT64 _PING6_PRIVATE_DATA::RttMin

Referenced by Ping6(), and Ping6OnEchoReplyReceived().

UINT64 _PING6_PRIVATE_DATA::RttSum

Referenced by Ping6(), and Ping6OnEchoReplyReceived().

UINT16 _PING6_PRIVATE_DATA::RxCount

Referenced by Ping6(), and Ping6OnEchoReplyReceived().

EFI_IP6_COMPLETION_TOKEN _PING6_PRIVATE_DATA::RxToken

Referenced by Ping6(), and Ping6OnEchoReplyReceived().

UINT32 _PING6_PRIVATE_DATA::SendNum

Referenced by Ping6(), Ping6OnEchoReplyReceived(), and Ping6OnTimerRoutine().

UINT32 _PING6_PRIVATE_DATA::SequenceNum

EFI_IPv6_ADDRESS _PING6_PRIVATE_DATA::SrcAddress

Referenced by Ping6().

EFI_STATUS _PING6_PRIVATE_DATA::Status

Referenced by Ping6(), Ping6OnEchoReplyReceived(), and Ping6OnTimerRoutine().

EFI_EVENT _PING6_PRIVATE_DATA::Timer

Referenced by Ping6().

UINT16 _PING6_PRIVATE_DATA::TxCount

Referenced by Ping6(), and Ping6OnTimerRoutine().

LIST_ENTRY _PING6_PRIVATE_DATA::TxList

Referenced by Ping6(), and Ping6OnTimerRoutine().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_PXEBC_PRIVATE_DATA Struct Reference

Data Fields

UINT32	Signature
EFI_HANDLE	Controller
EFI_HANDLE	Image
PXEBC_PRIVATE_PROTOCOL	Id
EFI_SIMPLE_NETWORK_PROTOCOL *	Snp
PXEBC_VIRTUAL_NIC *	Ip4Nic
PXEBC_VIRTUAL_NIC *	Ip6Nic
EFI_HANDLE	ArpChild
EFI_HANDLE	Ip4Child
EFI_HANDLE	Dhcp4Child
EFI_HANDLE	Mtftp4Child
EFI_HANDLE	Udp4ReadChild
EFI_HANDLE	Udp4WriteChild
EFI_ARP_PROTOCOL *	Arp
EFI_IP4_PROTOCOL *	Ip4
EFI_IP4_CONFIG2_PROTOCOL *	Ip4Config2
EFI_DHCP4_PROTOCOL *	Dhcp4
EFI_MTFTP4_PROTOCOL *	Mtftp4
EFI_UDP4_PROTOCOL *	Udp4Read
EFI_UDP4_PROTOCOL *	Udp4Write
EFI_HANDLE	Ip6Child
EFI_HANDLE	Dhcp6Child
EFI_HANDLE	Mtftp6Child
EFI_HANDLE	Udp6ReadChild
EFI_HANDLE	Udp6WriteChild
EFI_IP6_PROTOCOL *	Ip6
EFI_IP6_CONFIG_PROTOCOL *	Ip6Cfg
EFI_DHCP6_PROTOCOL *	Dhcp6
EFI_MTFTP6_PROTOCOL *	Mtftp6
EFI_UDP6_PROTOCOL *	Udp6Read
EFI_UDP6_PROTOCOL *	Udp6Write
EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL *	Nii
EFI_PXE_BASE_CODE_PROTOCOL	PxeBc
EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL	LoadFileCallback
EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL *	PxeBcCallback
EFI_DEVICE_PATH_PROTOCOL *	DevicePath
EFI_PXE_BASE_CODE_MODE	Mode
EFI_PXE_BASE_CODE_FUNCTION	Function
UINT32	Ip6Policy
UINT32	SolicitTimes
UINT64	ElapsedTime
EFI_UDP4_CONFIG_DATA	Udp4CfgData
EFI_UDP6_CONFIG_DATA	Udp6CfgData
EFI_IP4_CONFIG_DATA	Ip4CfgData
EFI_IP6_CONFIG_DATA	Ip6CfgData
EFI_EVENT	UdpTimeOutEvent
EFI_EVENT	ArpUpdateEvent
EFI_IP4_COMPLETION_TOKEN	IcmpToken

EFI_IP6_COMPLETION_TOKEN	Icmp6Token
BOOLEAN	IsAddressOk
BOOLEAN	IsOfferSorted
BOOLEAN	IsProxyRecved
BOOLEAN	IsDoDiscover
EFI_IP_ADDRESS	TmpStationIp
EFI_IP_ADDRESS	StationIp
EFI_IP_ADDRESS	SubnetMask
EFI_IP_ADDRESS	GatewayIp
EFI_IP_ADDRESS	ServerIp
UINT16	CurSrcPort
UINT32	IaId
UINT32	Ip4MaxPacketSize
UINT32	Ip6MaxPacketSize
UINT8 *	BootFileName
UINTN	BootFileSize
UINTN	BlockSize
PXEBC_DHCP_PACKET_CACHE	ProxyOffer
PXEBC_DHCP_PACKET_CACHE	DhcpAck
PXEBC_DHCP_PACKET_CACHE	PxeReply
EFI_DHCP6_PACKET *	Dhcp6Request
EFI_DHCP4_PACKET	SeedPacket
UINT32	SelectIndex
UINT32	SelectProxyType
PXEBC_DHCP_PACKET_CACHE	OfferBuffer [PXEBC_OFFER_MAX_NUM]
UINT32	OfferNum
UINT32	OfferCount [PxeOfferTypeMax]
UINT32	OfferIndex [PxeOfferTypeMax][PXEBC_OFFER_MAX_NUM]

Field Documentation

EFI_ARP_PROTOCOL* _PXEBC_PRIVATE_DATA::Arp

Referenced by EfiPxeBcArp(), EfiPxeBcSetStationIP(), and PxeBcArpCacheUpdate().

EFI_HANDLE _PXEBC_PRIVATE_DATA::ArpChild

EFI_EVENT _PXEBC_PRIVATE_DATA::ArpUpdateEvent

Referenced by EfiPxeBcStart(), and EfiPxeBcStop().

UINTN _PXEBC_PRIVATE_DATA::BlockSize

Referenced by EfiPxeBcStart().

UINT8* _PXEBC_PRIVATE_DATA::BootFileName

Referenced by EfiPxeBcStop().

UINTN _PXEBC_PRIVATE_DATA::BootFileSize

Referenced by EfiPxeBcStop().

EFI_HANDLE _PXEBC_PRIVATE_DATA::Controller

Referenced by EfiPxeBcSetParameters(), EfiPxeLoadFile(), PxeBcStart(), and PxeBcStop().

UINT16 _PXEBC_PRIVATE_DATA::CurSrcPort

Referenced by EfiPxeBcStop(), and EfiPxeBcUdpWrite().

EFI_DEVICE_PATH_PROTOCOL* _PXEBC_PRIVATE_DATA::DevicePath

Referenced by PxeBcStart().

EFI_DHCP4_PROTOCOL* _PXEBC_PRIVATE_DATA::Dhcp4

Referenced by EfiPxeBcDhcp(), and EfiPxeBcStop().

EFI_HANDLE _PXEBC_PRIVATE_DATA::Dhcp4Child

EFI_DHCP6_PROTOCOL* _PXEBC_PRIVATE_DATA::Dhcp6

Referenced by EfiPxeBcDhcp(), and EfiPxeBcStop().

EFI_HANDLE _PXEBC_PRIVATE_DATA::Dhcp6Child

EFI_DHCP6_PACKET* _PXEBC_PRIVATE_DATA::Dhcp6Request

Referenced by EfiPxeBcStop(), and PxeBcDhcp6CallBack().

PXEBC_DHCP_PACKET_CACHE _PXEBC_PRIVATE_DATA::DhcpAck

Referenced by EfiPxeBcStart().

UINT64 _PXEBC_PRIVATE_DATA::ElapsedTime

Referenced by EfiPxeBcDhcp(), and EfiPxeBcStop().

EFI_PXE_BASE_CODE_FUNCTION _PXEBC_PRIVATE_DATA::Function

Referenced by EfiPxeBcDhcp(), EfiPxeBcDiscover(), PxeBcDhcp4CallBack(), PxeBcDhcp6CallBack(), PxeBcMtftp4CheckPacket(), and PxeBcMtftp6CheckPacket().

EFI_IP_ADDRESS _PXEBC_PRIVATE_DATA::GatewayIp

Referenced by EfiPxeBcMtftp(), EfiPxeBcStop(), and EfiPxeBcUdpWrite().

UINT32 _PXEBC_PRIVATE_DATA::IaId

EFI_IP6_COMPLETION_TOKEN _PXEBC_PRIVATE_DATA::Icmp6Token

Referenced by EfiPxeBcStart(), EfiPxeBcStop(), PxeBcFlushStationIp(), and PxeBcIcmp6ErrorDpcHandle().

EFI_IP4_COMPLETION_TOKEN _PXEBC_PRIVATE_DATA::IcmpToken

Referenced by EfiPxeBcStart(), EfiPxeBcStop(), PxeBcFlushStationIp(), and PxeBcIcmpErrorDpcHandle().

PXEBC_PRIVATE_PROTOCOL _PXEBC_PRIVATE_DATA::Id

Referenced by PxeBcStart(), and PxeBcStop().

EFI_HANDLE _PXEBC_PRIVATE_DATA::Image

Referenced by PxeBcStart().

EFI_IP4_PROTOCOL* _PXEBC_PRIVATE_DATA::Ip4

Referenced by EfiPxeBcStart(), EfiPxeBcStop(), PxeBcFlushStationIp(), and PxeBcIcmpErrorDpcHandle().

EFI_IP4_CONFIG_DATA _PXEBC_PRIVATE_DATA::Ip4CfgData

Referenced by EfiPxeBcStop(), and PxeBcFlushStationIp().

EFI_HANDLE _PXEBC_PRIVATE_DATA::Ip4Child

EFI_IP4_CONFIG2_PROTOCOL* _PXEBC_PRIVATE_DATA::Ip4Config2

UINT32 _PXEBC_PRIVATE_DATA::Ip4MaxPacketSize

Referenced by EfiPxeBcStart().

PXEBC_VIRTUAL_NIC* _PXEBC_PRIVATE_DATA::Ip4Nic

Referenced by EfiPxeBcStart(), and PxeBcStop().

EFI_IP6_PROTOCOL* _PXEBC_PRIVATE_DATA::Ip6

Referenced by EfiPxeBcStart(), EfiPxeBcStop(), PxeBcFlushStationIp(), and PxeBcIcmp6ErrorDpcHandle().

EFI_IP6_CONFIG_PROTOCOL* _PXEBC_PRIVATE_DATA::Ip6Cfg

EFI_IP6_CONFIG_DATA _PXEBC_PRIVATE_DATA::Ip6CfgData

Referenced by EfiPxeBcStop(), and PxeBcFlushStationIp().

EFI_HANDLE _PXEBC_PRIVATE_DATA::Ip6Child

UINT32 _PXEBC_PRIVATE_DATA::Ip6MaxPacketSize

Referenced by EfiPxeBcStart().

PXEBC_VIRTUAL_NIC* _PXEBC_PRIVATE_DATA::Ip6Nic

Referenced by EfiPxeBcStart(), EfiPxeLoadFile(), and PxeBcStop().

UINT32 _PXEBC_PRIVATE_DATA::Ip6Policy

Referenced by EfiPxeBcStart().

BOOLEAN _PXEBC_PRIVATE_DATA::IsAddressOk

Referenced by EfiPxeBcArp(), EfiPxeBcDiscover(), EfiPxeBcSetStationIP(), EfiPxeBcStart(), and EfiPxeBcUdpWrite().

BOOLEAN _PXEBC_PRIVATE_DATA::IsDoDiscover

Referenced by EfiPxeBcDiscover().

BOOLEAN _PXEBC_PRIVATE_DATA::IsOfferSorted

Referenced by EfiPxeBcDhcp().

BOOLEAN _PXEBC_PRIVATE_DATA::IsProxyRecved

EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL _PXEBC_PRIVATE_DATA::LoadFileCallback

EFI_PXE_BASE_CODE_MODE _PXEBC_PRIVATE_DATA::Mode

Referenced by PxeBcIcmp6ErrorDpcHandle(), PxeBcIcmpErrorDpcHandle(), PxeBcMtftp4CheckPacket(), PxeBcMtftp6CheckPacket(), and PxeBcStart().

EFI_MTFTP4_PROTOCOL* _PXEBC_PRIVATE_DATA::Mtftp4

EFI_HANDLE _PXEBC_PRIVATE_DATA::Mtftp4Child

EFI_MTFTP6_PROTOCOL* _PXEBC_PRIVATE_DATA::Mtftp6

EFI_HANDLE _PXEBC_PRIVATE_DATA::Mtftp6Child

EFI_NETWORK_INTERFACE_IDENTIFIER_PROTOCOL* _PXEBC_PRIVATE_DATA::Nii

Referenced by PxeBcStart().

PXEBC_DHCP_PACKET_CACHE _PXEBC_PRIVATE_DATA::OfferBuffer[PXEBC_OFFER_MAX_NUM]

Referenced by EfiPxeBcStart(), PxeBcDhcp4CallBack(), and PxeBcDhcp6CallBack().

UINT32 _PXEBC_PRIVATE_DATA::OfferCount[PxeOfferTypeMax]

UINT32 _PXEBC_PRIVATE_DATA::OfferIndex[PxeOfferTypeMax][PXEBC_OFFER_MAX_NUM]

UINT32 _PXEBC_PRIVATE_DATA::OfferNum

Referenced by PxeBcDhcp4CallBack(), and PxeBcDhcp6CallBack().

PXEBC_DHCP_PACKET_CACHE _PXEBC_PRIVATE_DATA::ProxyOffer

Referenced by EfiPxeBcStart().

EFI_PXE_BASE_CODE_PROTOCOL _PXEBC_PRIVATE_DATA::PxeBc

Referenced by EfiPxeBcArp(), EfiPxeBcDhcp(), EfiPxeBcDiscover(), EfiPxeBcMtftp(), EfiPxeBcSetIpFilter(), EfiPxeBcSetPackets(), EfiPxeBcSetParameters(), EfiPxeBcSetStationIP(), EfiPxeBcStart(), EfiPxeBcStop(), EfiPxeBcUdpRead(), EfiPxeBcUdpWrite(), EfiPxeLoadFile(), PxeBcArpCacheUpdate(), PxeBcDhcp4CallBack(), PxeBcDhcp6CallBack(), PxeBcFlushStationIp(), PxeBcStart(), and PxeBcStop().

EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL* _PXEBC_PRIVATE_DATA::PxeBcCallback

Referenced by EfiPxeBcSetParameters(), PxeBcDhcp4CallBack(), PxeBcDhcp6CallBack(), PxeBcMtftp4CheckPacket(), and PxeBcMtftp6CheckPacket().

PXEBC_DHCP_PACKET_CACHE _PXEBC_PRIVATE_DATA::PxeReply

Referenced by EfiPxeBcDiscover(), and EfiPxeBcStart().

EFI_DHCP4_PACKET _PXEBC_PRIVATE_DATA::SeedPacket

Referenced by EfiPxeBcStart().

UINT32 _PXEBC_PRIVATE_DATA::SelectIndex

Referenced by PxeBcDhcp4CallBack(), and PxeBcDhcp6CallBack().

UINT32 _PXEBC_PRIVATE_DATA::SelectProxyType

EFI_IP_ADDRESS _PXEBC_PRIVATE_DATA::ServerIp

Referenced by EfiPxeBcDiscover(), and EfiPxeBcStop().

UINT32 _PXEBC_PRIVATE_DATA::Signature

Referenced by PxeBcStart().

EFI_SIMPLE_NETWORK_PROTOCOL* _PXEBC_PRIVATE_DATA::Snp

Referenced by PxeBcStart().

UINT32 _PXEBC_PRIVATE_DATA::SolicitTimes

Referenced by EfiPxeBcDhcp(), EfiPxeBcStop(), and PxeBcDhcp6CallBack().

EFI_IP_ADDRESS _PXEBC_PRIVATE_DATA::StationIp

Referenced by EfiPxeBcMtftp(), EfiPxeBcSetStationIP(), EfiPxeBcStop(), and EfiPxeBcUdpWrite().

EFI_IP_ADDRESS _PXEBC_PRIVATE_DATA::SubnetMask

Referenced by EfiPxeBcMtftp(), EfiPxeBcSetStationIP(), EfiPxeBcStop(), and EfiPxeBcUdpWrite().

EFI_IP_ADDRESS _PXEBC_PRIVATE_DATA::TmpStationIp

EFI_UDP4_CONFIG_DATA _PXEBC_PRIVATE_DATA::Udp4CfgData

Referenced by EfiPxeBcDhcp(), EfiPxeBcDiscover(), EfiPxeBcMtftp(), EfiPxeBcSetIpFilter(), EfiPxeBcStart(), EfiPxeBcStop(), and PxeBcFlushStationIp().

EFI_UDP4_PROTOCOL* _PXEBC_PRIVATE_DATA::Udp4Read

Referenced by EfiPxeBcDhcp(), EfiPxeBcDiscover(), EfiPxeBcMtftp(), EfiPxeBcSetIpFilter(), EfiPxeBcStart(), EfiPxeBcStop(), and EfiPxeBcUdpRead().

EFI_HANDLE _PXEBC_PRIVATE_DATA::Udp4ReadChild

EFI_UDP4_PROTOCOL* _PXEBC_PRIVATE_DATA::Udp4Write

Referenced by EfiPxeBcStop(), and EfiPxeBcUdpWrite().

EFI_HANDLE _PXEBC_PRIVATE_DATA::Udp4WriteChild

EFI_UDP6_CONFIG_DATA _PXEBC_PRIVATE_DATA::Udp6CfgData

Referenced by EfiPxeBcDhcp(), EfiPxeBcDiscover(), EfiPxeBcMtftp(), EfiPxeBcSetIpFilter(), EfiPxeBcStart(), EfiPxeBcStop(), and PxeBcFlushStationIp().

EFI_UDP6_PROTOCOL* _PXEBC_PRIVATE_DATA::Udp6Read

Referenced by EfiPxeBcDhcp(), EfiPxeBcDiscover(), EfiPxeBcMtftp(), EfiPxeBcSetIpFilter(), EfiPxeBcStart(), EfiPxeBcStop(), and EfiPxeBcUdpRead().

EFI_HANDLE _PXEBC_PRIVATE_DATA::Udp6ReadChild

EFI_UDP6_PROTOCOL* _PXEBC_PRIVATE_DATA::Udp6Write

Referenced by EfiPxeBcStop(), and EfiPxeBcUdpWrite().

EFI_HANDLE _PXEBC_PRIVATE_DATA::Udp6WriteChild

EFI_EVENT _PXEBC_PRIVATE_DATA::UdpTimeOutEvent

Referenced by EfiPxeBcStart(), EfiPxeBcStop(), EfiPxeBcUdpRead(), and EfiPxeBcUdpWrite().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

_PXEBC_PRIVATE_PROTOCOL Struct Reference

Data Fields

UINT64 Reserved

Field Documentation

UINT64 _PXEBC_PRIVATE_PROTOCOL::Reserved

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_PXEBC_VIRTUAL_NIC Struct Reference

Data Fields

UINT32	Signature
EFI_HANDLE	Controller
EFI_LOAD_FILE_PROTOCOL	LoadFile
EFI_DEVICE_PATH_PROTOCOL *	DevicePath
PXEBC_PRIVATE_DATA *	Private

Field Documentation

EFI_HANDLE _PXEBC_VIRTUAL_NIC::Controller

EFI_DEVICE_PATH_PROTOCOL* _PXEBC_VIRTUAL_NIC::DevicePath

EFI_LOAD_FILE_PROTOCOL _PXEBC_VIRTUAL_NIC::LoadFile

PXEBC_PRIVATE_DATA* _PXEBC_VIRTUAL_NIC::Private

Referenced by EfiPxeLoadFile(), and PxeBcStop().

UINT32 _PXEBC_VIRTUAL_NIC::Signature

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

_SCSI_COMMAND Struct Reference

Data Fields

UINT8 OpCode
UINT8 Flags
UINT16 Reserved
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT8 Lun [8]
UINT32 InitiatorTaskTag
UINT32 ExpDataXferLength
UINT32 CmdSN
UINT32 ExpStatSN
UINT8 Cdb [16]

Detailed Description

SCSI Command

Field Documentation

UINT8 _SCSI_COMMAND::Cdb[16]

Referenced by IScsiNewScsiCmdPdu().

UINT32 _SCSI_COMMAND::CmdSN

Referenced by IScsiNewScsiCmdPdu().

UINT8 _SCSI_COMMAND::DataSegmentLength[3]

UINT32 _SCSI_COMMAND::ExpDataXferLength

Referenced by IScsiNewScsiCmdPdu().

UINT32 _SCSI_COMMAND::ExpStatSN

Referenced by IScsiNewScsiCmdPdu().

UINT8 _SCSI_COMMAND::Flags

UINT32 _SCSI_COMMAND::InitiatorTaskTag

Referenced by IScsiNewScsiCmdPdu().

UINT8 _SCSI_COMMAND::Lun[8]

Referenced by IScsiNewScsiCmdPdu().

UINT8 _SCSI_COMMAND::OpCode

UINT16 _SCSI_COMMAND::Reserved

UINT8 _SCSI_COMMAND::TotalAHSLength

Referenced by IScsiNewScsiCmdPdu().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_SCSI_RESPONSE Struct Reference

Data Fields

UINT8 OpCode
UINT8 Flags
UINT8 Response
UINT8 Status
UINT8 TotalAHSLength
UINT8 DataSegmentLength [3]
UINT8 Reserved [8]
UINT32 InitiatorTaskTag
UINT32 SNACKTag
UINT32 StatSN
UINT32 ExpCmdSN
UINT32 MaxCmdSN
UINT32 ExpDataSN
UINT32 BiReadResidualCount
UINT32 ResidualCount

Detailed Description

SCSI Response

Field Documentation

UINT32 _SCSI_RESPONSE::BiReadResidualCount

Referenced by IScsiOnScsiRspRcvd().

UINT8 _SCSI_RESPONSE::DataSegmentLength[3]

UINT32 _SCSI_RESPONSE::ExpCmdSN

Referenced by IScsiOnScsiRspRcvd().

UINT32 _SCSI_RESPONSE::ExpDataSN

UINT8 _SCSI_RESPONSE::Flags

UINT32 _SCSI_RESPONSE::InitiatorTaskTag

Referenced by IScsiOnScsiRspRcvd().

UINT32 _SCSI_RESPONSE::MaxCmdSN

Referenced by IScsiOnScsiRspRcvd().

UINT8 _SCSI_RESPONSE::OpCode

UINT8 _SCSI_RESPONSE::Reserved[8]

UINT32 _SCSI_RESPONSE::ResidualCount

Referenced by IScsiOnScsiRspRcvd().

UINT8 _SCSI_RESPONSE::Response

Referenced by IScsiOnScsiRspRcvd().

UINT32 _SCSI_RESPONSE::SNACKTag

UINT32 _SCSI_RESPONSE::StatSN

Referenced by IScsiOnScsiRspRcvd().

UINT8 _SCSI_RESPONSE::Status

Referenced by IScsiOnScsiRspRcvd().

UINT8 _SCSI_RESPONSE::TotalAHSLength

_SOCK_BUFFER Struct Reference

Data Fields

UINT32	HighWater The buffersize upper limit of sock_buffer.
UINT32	LowWater The low warter mark of sock_buffer.
NET_BUF_QUEUE *	DataQueue The queue to buffer data.

Detailed Description

The buffer structure of rcvd data and send data used by socket.

Field Documentation

NET_BUF_QUEUE* _SOCK_BUFFER::DataQueue
The queue to buffer data.

Referenced by SockCreate(), and SockGetFreeSpace().

UINT32 _SOCK_BUFFER::HighWater
The buffersize upper limit of sock_buffer.

Referenced by SockCreate(), and SockGetFreeSpace().

UINT32 _SOCK_BUFFER::LowWater
The low warter mark of sock_buffer.

Referenced by SockCreate().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_SOCK_COMPLETION_TOKEN Struct Reference

Data Fields

EFI_EVENT Event
: The event to be issued.

EFI_STATUS Status
: The status to be issued.

Detailed Description

Socket completion token

Field Documentation

EFI_EVENT _SOCK_COMPLETION_TOKEN::Event

The event to be issued.

Referenced by SockRcv(), SockSend(), and SockTokenExistedInList().

EFI_STATUS _SOCK_COMPLETION_TOKEN::Status

The status to be issued.

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

_SOCK_INIT_DATA Struct Reference

Data Fields

SOCK_TYPE	Type
UINT8	State
SOCKET *	Parent The parent of this socket.
UINT32	BackLog The connection limit for listening socket.
UINT32	SndBufferSize The high warter mark of send buffer.
UINT32	RcvBufferSize The high warter mark of receive buffer.
UINT8	IpVersion
VOID *	Protocol
SOCK_CREATE_CALLBACK	CreateCallback Callback after created.
SOCK_DESTROY_CALLBACK	DestroyCallback Callback before destroied.
VOID *	Context The context of the callback.
VOID *	ProtoData
UINT32	DataSize
SOCK_PROTO_HANDLER	ProtoHandler The handler of protocol for socket request.
EFI_HANDLE	DriverBinding The driver binding handle.

Detailed Description

The initialize data for create a new socket.

Field Documentation

UINT32 _SOCK_INIT_DATA::BackLog
The connection limit for listening socket.

Referenced by SockClone().

VOID* _SOCK_INIT_DATA::Context
The context of the callback.

Referenced by SockClone().

SOCK_CREATE_CALLBACK _SOCK_INIT_DATA::CreateCallback

Callback after created.

Referenced by SockClone().

UINT32 _SOCK_INIT_DATA::DataSize

Referenced by SockClone(), and TcpServiceBindingCreateChild().

SOCK_DESTROY_CALLBACK _SOCK_INIT_DATA::DestroyCallback
Callback before destroied.

Referenced by SockClone().

EFI_HANDLE _SOCK_INIT_DATA::DriverBinding
The driver binding handle.

Referenced by SockClone(), and TcpServiceBindingCreateChild().

UINT8 _SOCK_INIT_DATA::IpVersion

Referenced by SockClone(), and TcpServiceBindingCreateChild().

SOCKET* _SOCK_INIT_DATA::Parent
The parent of this socket.

Referenced by SockClone().

VOID* _SOCK_INIT_DATA::Protocol
The pointer to protocol function template wanted to install on socket

Referenced by SockClone(), and TcpServiceBindingCreateChild().

VOID* _SOCK_INIT_DATA::ProtoData

Referenced by SockClone(), and TcpServiceBindingCreateChild().

SOCK_PROTO_HANDLER _SOCK_INIT_DATA::ProtoHandler
The handler of protocol for socket request.

Referenced by SockClone().

UINT32 _SOCK_INIT_DATA::RcvBufferSize
The high warter mark of receive buffer.

Referenced by SockClone().

UINT32 _SOCK_INIT_DATA::SndBufferSize
The high warter mark of send buffer.

Referenced by SockClone().

UINT8 _SOCK_INIT_DATA::State

Referenced by SockClone().

SOCK_TYPE _SOCK_INIT_DATA::Type

Referenced by SockClone().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_SOCK_IO_TOKEN Struct Reference

Data Fields

SOCK_COMPLETION_TOKEN Token
SOCK_IO_DATA Packet

Detailed Description

The application token with data packet

Field Documentation

SOCK_IO_DATA _SOCK_IO_TOKEN::Packet

Referenced by SockProcessSndToken(), and SockSend().

SOCK_COMPLETION_TOKEN _SOCK_IO_TOKEN::Token

Referenced by SockRcv(), and SockSend().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

_SOCK_TOKEN Struct Reference

Data Fields

LIST_ENTRY	TokenList The entry to add in the token list.
SOCK_COMPLETION_TOKEN *	Token The application's token.
UINT32	RemainDataLen Unprocessed data length.
SOCKET *	Sock

Detailed Description

The token structure buffered in socket layer.

Field Documentation

UINT32 _SOCK_TOKEN::RemainDataLen
Unprocessed data length.

Referenced by SockBufferToken(), and SockDataSent().

SOCKET* _SOCK_TOKEN::Sock
The poninter to the socket this token belongs to

Referenced by SockBufferToken().

SOCK_COMPLETION_TOKEN* _SOCK_TOKEN::Token
The application's token.

Referenced by SockBufferToken(), SockDataSent(), SockFlushPendingToken(), SockProcessSndToken(), SockTokenExistedInList(), SockWakeListenToken(), and SockWakeRcvToken().

LIST_ENTRY _SOCK_TOKEN::TokenList
The entry to add in the token list.

Referenced by SockBufferToken(), SockDataSent(), SockFlushPendingToken(), SockProcessSndToken(), SockSend(), SockWakeListenToken(), and SockWakeRcvToken().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Data Structures
- Data Fields

_TCP4_MODE_DATA Struct Reference

Data Fields

EFI_TCP4_CONNECTION_STATE *	Tcp4State
EFI_TCP4_CONFIG_DATA *	Tcp4ConfigData
EFI_IP4_MODE_DATA *	Ip4ModeData
EFI_MANAGED_NETWORK_CONFIG_DATA *	MnpConfigData
EFI_SIMPLE_NETWORK_MODE *	SnpModeData

Field Documentation

EFI_IP4_MODE_DATA* _TCP4_MODE_DATA::Ip4ModeData

Referenced by Tcp4GetModeData().

EFI_MANAGED_NETWORK_CONFIG_DATA* _TCP4_MODE_DATA::MnpConfigData

Referenced by Tcp4GetModeData().

EFI_SIMPLE_NETWORK_MODE* _TCP4_MODE_DATA::SnpModeData

Referenced by Tcp4GetModeData().

EFI_TCP4_CONFIG_DATA* _TCP4_MODE_DATA::Tcp4ConfigData

Referenced by Tcp4GetModeData().

EFI_TCP4_CONNECTION_STATE* _TCP4_MODE_DATA::Tcp4State

Referenced by Tcp4GetModeData().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_TCP4_ROUTE_INFO Struct Reference

Data Fields

BOOLEAN DeleteRoute
EFI_IPv4_ADDRESS * SubnetAddress
EFI_IPv4_ADDRESS * SubnetMask
EFI_IPv4_ADDRESS * GatewayAddress

Field Documentation

BOOLEAN _TCP4_ROUTE_INFO::DeleteRoute

Referenced by Tcp4Routes().

EFI_IPv4_ADDRESS* _TCP4_ROUTE_INFO::GatewayAddress

Referenced by Tcp4Routes().

EFI_IPv4_ADDRESS* _TCP4_ROUTE_INFO::SubnetAddress

Referenced by Tcp4Routes().

EFI_IPv4_ADDRESS* _TCP4_ROUTE_INFO::SubnetMask

Referenced by Tcp4Routes().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_TCP6_MODE_DATA Struct Reference

Data Fields

EFI_TCP6_CONNECTION_STATE *	Tcp6State
EFI_TCP6_CONFIG_DATA *	Tcp6ConfigData
EFI_IP6_MODE_DATA *	Ip6ModeData
EFI_MANAGED_NETWORK_CONFIG_DATA *	MnpConfigData
EFI_SIMPLE_NETWORK_MODE *	SnpModeData

Field Documentation

EFI_IP6_MODE_DATA* _TCP6_MODE_DATA::Ip6ModeData

Referenced by Tcp6GetModeData().

EFI_MANAGED_NETWORK_CONFIG_DATA* _TCP6_MODE_DATA::MnpConfigData

Referenced by Tcp6GetModeData().

EFI_SIMPLE_NETWORK_MODE* _TCP6_MODE_DATA::SnpModeData

Referenced by Tcp6GetModeData().

EFI_TCP6_CONFIG_DATA* _TCP6_MODE_DATA::Tcp6ConfigData

Referenced by Tcp6GetModeData().

EFI_TCP6_CONNECTION_STATE* _TCP6_MODE_DATA::Tcp6State

Referenced by Tcp6GetModeData().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

_TCP_CONTROL_BLOCK Struct Reference

Data Fields

LIST_ENTRY List
Back and forward link entry.

TCP_CB * Parent
The parent TCP_CB structure.

SOCKET * Sk
The socket it controled.

TCP_PEER LocalEnd
Local endpoint.

TCP_PEER RemoteEnd
Remote endpoint.

LIST_ENTRY SndQue
Retxmission queue.

LIST_ENTRY RcvQue
Reassemble queue.

UINT32 CtrlFlag
Control flags, such as NO_NAGLE.

INT32 Error
Soft error status, such as TCP_CONNECT_RESET.

UINT8 State
TCP state, such as SYN_SENT, LISTEN.

UINT8 DelayedAck
Number of delayed ACKs.

UINT16 HeadSum

TCP_SEQNO Iss
Initial Sending Sequence.

TCP_SEQNO SndUna
First unacknowledged data.

TCP_SEQNO SndNxt
Next data sequence to send.

TCP_SEQNO SndPsh
Send PUSH point.

TCP_SEQNO SndUp
Send urgent point.

UINT32 SndWnd
Window advertised by the remote peer.

UINT32 SndWndMax
Max send window advertised by the peer.

TCP_SEQNO SndWl1
Seq number used for last window update.

TCP_SEQNO SndWl2
Ack no of last window update.

UINT16 SndMss
Max send segment size.

TCP_SEQNO RcvNxt
Next sequence no to receive.

UINT32 RcvWnd
Window advertised by the local peer.

TCP_SEQNO RcvWl2

TCP_SEQNO RcvUp
Urgent point;.
TCP_SEQNO Irs
Initial Receiving Sequence.
UINT16 RcvMss
Max receive segment size.
UINT16 EnabledTimer
Which timer is currently enabled.
UINT32 Timer [TCP_TIMER_NUMBER]
When the timer will expire.
INT32 NextExpire
Countdown offset for the nearest timer.
UINT32 Idle
How long the connection is in idle.
UINT32 ProbeTime
The time out value for current window prober.
BOOLEAN ProbeTimerOn
If TRUE, the probe time is on.
UINT8 SndWndScale
Wndscale received from the peer.
UINT8 RcvWndScale
Wndscale used to scale local buffer.
UINT32 TsRecent
TsRecent to echo to the remote peer.
UINT32 TsRecentAge
When this TsRecent is updated.
TCP_SEQNO RttSeq
The seq of measured segment now.
UINT32 RttMeasure
Currently measured RTT in heartbeats.
UINT32 SRtt
Smoothed RTT, scaled by 8.
UINT32 RttVar
RTT variance, scaled by 8.
UINT32 Rto
Current RTO, not scaled.
UINT32 CWnd
Sender's congestion window.
UINT32 Ssthresh
Slow start threshold.
TCP_SEQNO Recover
Recover point for NewReno.
UINT16 DupAck
Number of duplicate ACKs.
UINT8 CongestState
The current congestion state(RFC3782).
UINT8 LossTimes
Number of retxmit timeouts in a row.
TCP_SEQNO LossRecover
Recover point for retxmit.
UINT32 KeepAliveIdle
Idle time before sending first probe.
UINT32 KeepAlivePeriod
Interval for subsequent keep alive probe.

UINT8 MaxKeepAlive
Maxium keep alive probe times.
UINT8 KeepAliveProbes
The number of keep alive probe.
UINT16 MaxRexmit
The maxium number of retxmit before abort.
UINT32 FinWait2Timeout
The FIN_WAIT_2 timeout.
UINT32 TimeWaitTimeout
The TIME_WAIT timeout.
UINT32 ConnectTimeout
The connect establishment timeout.
BOOLEAN UseDefaultAddr
UINT8 Tos
UINT8 Ttl
EFI_IPv4_ADDRESS SubnetMask
BOOLEAN RemoteIpZero
RemoteEnd.Ip is ZERO when configured.
IP_IO_IP_INFO * IpInfo
Pointer reference to Ip used to send pkt.
UINT32 Tick
1 tick = 200ms

Detailed Description

TCP control block: it includes various states.

Field Documentation

UINT8 _TCP_CONTROL_BLOCK::CongestState
The current congestion state(RFC3782).

Referenced by TcpConfigurePcb(), and TcpInput().

UINT32 _TCP_CONTROL_BLOCK::ConnectTimeout
The connect establishment timeout.

Referenced by TcpConfigurePcb(), and TcpInput().

UINT32 _TCP_CONTROL_BLOCK::CtrlFlag
Control flags, such as NO_NAGLE.

Referenced by TcpConfigurePcb(), TcpDispatcher(), TcpInput(), and TcpTickingDpc().

UINT32 _TCP_CONTROL_BLOCK::CWnd
Sender's congestion window.

Referenced by TcpConfigurePcb(), and TcpInput().

UINT8 _TCP_CONTROL_BLOCK::DelayedAck
Number of delayed ACKs.

Referenced by TcpTickingDpc().

UINT16 _TCP_CONTROL_BLOCK::DupAck
Number of duplicate ACKs.

Referenced by TcpInput().

UINT16 _TCP_CONTROL_BLOCK::EnabledTimer
Which timer is currently enabled.

Referenced by TcpTickingDpc().

INT32 _TCP_CONTROL_BLOCK::Error
Soft error status, such as TCP_CONNECT_RESET.

UINT32 _TCP_CONTROL_BLOCK::FinWait2Timeout
The FIN_WAIT_2 timeout.

Referenced by TcpConfigurePcb(), and TcpInput().

UINT16 _TCP_CONTROL_BLOCK::HeadSum
Checksum of the fixed parts of pesudo header: Src IP, Dst IP, 0, Protocol, do not include the TCP length.

UINT32 _TCP_CONTROL_BLOCK::Idle
How long the connection is in idle.

Referenced by TcpInput(), and TcpTickingDpc().

IP_IO_IP_INFO* _TCP_CONTROL_BLOCK::IpInfo
Pointer reference to Ip used to send pkt.

Referenced by TcpAttachPcb(), TcpConfigurePcb(), TcpDetachPcb(), TcpInput(), and TcpTickingDpc().

TCP_SEQNO _TCP_CONTROL_BLOCK::Irs
Initial Receiving Sequence.

TCP_SEQNO _TCP_CONTROL_BLOCK::Iss

Initial Sending Sequence.

Referenced by TcpInput().

UINT32 _TCP_CONTROL_BLOCK::KeepAliveIdle
Idle time before sending first probe.

Referenced by TcpConfigurePcb().

UINT32 _TCP_CONTROL_BLOCK::KeepAlivePeriod
Interval for subsequent keep alive probe.

Referenced by TcpConfigurePcb().

UINT8 _TCP_CONTROL_BLOCK::KeepAliveProbes
The number of keep alive probe.

LIST_ENTRY _TCP_CONTROL_BLOCK::List
Back and forward link entry.

Referenced by TcpAttachPcb(), TcpCloneTcb(), and TcpLocateTcb().

TCP_PEER _TCP_CONTROL_BLOCK::LocalEnd
Local endpoint.

Referenced by TcpConfigurePcb(), TcpFindTcbByPeer(), TcpInput(), TcpInsertTcb(), TcpInstallDevicePath(), TcpLocateListenTcb(), and TcpLocateTcb().

TCP_SEQNO _TCP_CONTROL_BLOCK::LossRecover
Recover point for retxmit.

Referenced by TcpInput().

UINT8 _TCP_CONTROL_BLOCK::LossTimes
Number of retxmit timeouts in a row.

UINT8 _TCP_CONTROL_BLOCK::MaxKeepAlive
Maxium keep alive probe times.

Referenced by TcpConfigurePcb().

UINT16 _TCP_CONTROL_BLOCK::MaxRexmit
The maxium number of retxmit before abort.

Referenced by TcpConfigurePcb().

INT32 _TCP_CONTROL_BLOCK::NextExpire
Countdown offset for the nearest timer.

Referenced by TcpTickingDpc().

TCP_CB* _TCP_CONTROL_BLOCK::Parent
The parent TCP_CB structure.

Referenced by TcpInput().

UINT32 _TCP_CONTROL_BLOCK::ProbeTime
The time out value for current window prober.

BOOLEAN _TCP_CONTROL_BLOCK::ProbeTimerOn
If TRUE, the probe time is on.

Referenced by TcpInput().

UINT16 _TCP_CONTROL_BLOCK::RcvMss
Max receive segment size.

Referenced by TcpConfigurePcb().

TCP_SEQNO _TCP_CONTROL_BLOCK::RcvNxt
Next sequence no to receive.

LIST_ENTRY _TCP_CONTROL_BLOCK::RcvQue
Reassemble queue.

Referenced by TcpAttachPcb(), TcpCloneTcb(), TcpConfigurePcb(), and TcpInput().

TCP_SEQNO _TCP_CONTROL_BLOCK::RcvUp
Urgent point;.

Referenced by TcpInput().

TCP_SEQNO _TCP_CONTROL_BLOCK::RcvWl2
The RcvNxt (or ACK) of last window update. It is necessary because of delayed ACK.

Referenced by TcpInput().

UINT32 _TCP_CONTROL_BLOCK::RcvWnd
Window advertised by the local peer.

UINT8 _TCP_CONTROL_BLOCK::RcvWndScale
Wndscale used to scale local buffer.

TCP_SEQNO _TCP_CONTROL_BLOCK::Recover
Recover point for NewReno.

Referenced by TcpInput().

TCP_PEER _TCP_CONTROL_BLOCK::RemoteEnd
Remote endpoint.

Referenced by TcpConfigurePcb(), TcpInput(), TcpInsertTcb(), TcpInstallDevicePath(), TcpLocateListenTcb(), and TcpLocateTcb().

BOOLEAN _TCP_CONTROL_BLOCK::RemoteIpZero
RemoteEnd.Ip is ZERO when configured.

Referenced by TcpConfigurePcb().

UINT32 _TCP_CONTROL_BLOCK::Rto
Current RTO, not scaled.

Referenced by TcpConfigurePcb(), and TcpInput().

UINT32 _TCP_CONTROL_BLOCK::RttMeasure
Currently measured RTT in heartbeats.

Referenced by TcpInput(), and TcpTickingDpc().

TCP_SEQNO _TCP_CONTROL_BLOCK::RttSeq
The seq of measured segment now.

UINT32 _TCP_CONTROL_BLOCK::RttVar
RTT variance, scaled by 8.

SOCKET* _TCP_CONTROL_BLOCK::Sk
The socket it controled.

Referenced by TcpAttachPcb(), TcpCloneTcb(), TcpDispatcher(), TcpFindTcbByPeer(), TcpIcmpInput(), TcpInput(), TcpLocateListenTcb(), and TcpLocateTcb().

UINT16 _TCP_CONTROL_BLOCK::SndMss
Max send segment size.

Referenced by TcpConfigurePcb(), and TcpInput().

TCP_SEQNO _TCP_CONTROL_BLOCK::SndNxt
Next data sequence to send.

Referenced by TcpIcmpInput(), and TcpInput().

TCP_SEQNO _TCP_CONTROL_BLOCK::SndPsh
Send PUSH point.

Referenced by TcpDispatcher().

LIST_ENTRY _TCP_CONTROL_BLOCK::SndQue
Retxmission queue.

Referenced by TcpAttachPcb(), TcpCloneTcb(), and TcpConfigurePcb().

TCP_SEQNO _TCP_CONTROL_BLOCK::SndUna
First unacknowledged data.

Referenced by TcpIcmpInput(), and TcpInput().

TCP_SEQNO _TCP_CONTROL_BLOCK::SndUp
Send urgent point.

Referenced by TcpDispatcher(), and TcpInput().

TCP_SEQNO _TCP_CONTROL_BLOCK::SndWl1
Seq number used for last window update.

Referenced by TcpInput().

TCP_SEQNO _TCP_CONTROL_BLOCK::SndWl2
Ack no of last window update.

Referenced by TcpInput().

UINT32 _TCP_CONTROL_BLOCK::SndWnd
Window advertised by the remote peer.

Referenced by TcpInput().

UINT32 _TCP_CONTROL_BLOCK::SndWndMax
Max send window advertised by the peer.

Referenced by TcpInput().

UINT8 _TCP_CONTROL_BLOCK::SndWndScale
Wndscale received from the peer.

Referenced by TcpInput().

UINT32 _TCP_CONTROL_BLOCK::SRtt
Smoothed RTT, scaled by 8.

Referenced by TcpConfigurePcb().

UINT32 _TCP_CONTROL_BLOCK::Ssthresh
Slow start threshold.

Referenced by TcpConfigurePcb(), and TcpInput().

UINT8 _TCP_CONTROL_BLOCK::State
TCP state, such as SYN_SENT, LISTEN.

Referenced by TcpAttachPcb(), TcpConfigurePcb(), TcpIcmpInput(), TcpInput(), and TcpTickingDpc().

EFI_IPv4_ADDRESS _TCP_CONTROL_BLOCK::SubnetMask

Referenced by TcpConfigurePcb(), and TcpInstallDevicePath().

UINT32 _TCP_CONTROL_BLOCK::Tick
1 tick = 200ms

Referenced by TcpConfigurePcb(), TcpInput(), and TcpTickingDpc().

UINT32 _TCP_CONTROL_BLOCK::Timer[TCP_TIMER_NUMBER]
When the timer will expire.

Referenced by TcpTickingDpc().

UINT32 _TCP_CONTROL_BLOCK::TimeWaitTimeout
The TIME_WAIT timeout.

Referenced by TcpConfigurePcb(), and TcpInput().

UINT8 _TCP_CONTROL_BLOCK::Tos

Referenced by TcpConfigurePcb().

UINT32 _TCP_CONTROL_BLOCK::TsRecent
TsRecent to echo to the remote peer.

Referenced by TcpInput().

UINT32 _TCP_CONTROL_BLOCK::TsRecentAge
When this TsRecent is updated.

Referenced by TcpInput().

UINT8 _TCP_CONTROL_BLOCK::Ttl

Referenced by TcpConfigurePcb().

BOOLEAN _TCP_CONTROL_BLOCK::UseDefaultAddr

Referenced by TcpConfigurePcb(), and TcpInstallDevicePath().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_TCP_HEARTBEAT_TIMER Struct Reference

Data Fields

EFI_EVENT TimerEvent
INTN RefCnt

Field Documentation

INTN _TCP_HEARTBEAT_TIMER::RefCnt

Referenced by TcpCreateTimer(), and TcpDestroyTimer().

EFI_EVENT _TCP_HEARTBEAT_TIMER::TimerEvent

Referenced by TcpCreateTimer(), and TcpDestroyTimer().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

_TCP_OPTION Struct Reference

Data Fields

UINT8 Flag
Flag such as TCP_OPTION_RCVD_MSS.

UINT8 WndScale
The WndScale received.

UINT16 Mss
The Mss received.

UINT32 TSVal
The TSVal field in a timestamp option.

UINT32 TSEcr
The TSEcr field in a timestamp option.

Detailed Description

The structure to store the parse option value. ParseOption only parses the options, doesn't process them.

Field Documentation

UINT8 _TCP_OPTION::Flag
Flag such as TCP_OPTION_RCVD_MSS.

Referenced by TcpInput().

UINT16 _TCP_OPTION::Mss
The Mss received.

UINT32 _TCP_OPTION::TSEcr
The TSEcr field in a timestamp option.

Referenced by TcpInput().

UINT32 _TCP_OPTION::TSVal
The TSVal field in a timestamp option.

Referenced by TcpInput().

UINT8 _TCP_OPTION::WndScale
The WndScale received.

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_TCP_PEER Struct Reference

Data Fields

EFI_IP_ADDRESS Ip
IP address, in network byte order.

TCP_PORTNO Port
Port number, in network byte order.

Detailed Description

Network endpoint, IP plus Port structure.

Field Documentation

EFI_IP_ADDRESS _TCP_PEER::Ip
IP address, in network byte order.

Referenced by TcpConfigurePcb(), TcpFindTcbByPeer(), TcpInput(), TcpInstallDevicePath(), TcpLocateListenTcb(), and TcpLocateTcb().

TCP_PORTNO _TCP_PEER::Port
Port number, in network byte order.

Referenced by TcpConfigurePcb(), TcpFindTcbByPeer(), TcpInput(), TcpInstallDevicePath(), TcpLocateListenTcb(), and TcpLocateTcb().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_TCP_PROTO_DATA Struct Reference

Data Fields

TCP_SERVICE_DATA * TcpService
TCP_CB * TcpPcb

Field Documentation

TCP_CB* _TCP_PROTO_DATA::TcpPcb

Referenced by TcpAttachPcb(), TcpConfigurePcb(), TcpDetachPcb(), TcpDispatcher(), TcpInstallDevicePath(), and TcpServiceBindingCreateChild().

TCP_SERVICE_DATA* _TCP_PROTO_DATA::TcpService

Referenced by Tcp6RefreshNeighbor(), TcpAttachPcb(), TcpDetachPcb(), TcpDispatcher(), TcpGetRcvMss(), TcpInstallDevicePath(), TcpSendIpPacket(), and TcpServiceBindingCreateChild().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

_TCP_RSV_DATA Struct Reference

Data Fields

UINT32 UrgLen

Detailed Description

Reserved data to access the NET_BUF delivered by TCP driver.

Field Documentation

UINT32 _TCP_RSV_DATA::UrgLen

Referenced by SockTcpDataToRcv().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_TCP_SEG Struct Reference

Data Fields

TCP_SEQNO Seq
Starting sequence number.

TCP_SEQNO End
The sequence of the last byte + 1, include SYN/FIN. End-Seq = SEG.LEN.

TCP_SEQNO Ack
ACK field in the segment.

UINT8 Flag
TCP header flags.

UINT16 Urg
Valid if URG flag is set.

UINT32 Wnd
TCP window size field.

Detailed Description

TCP segmentation data.

Field Documentation

TCP_SEQNO _TCP_SEG::Ack
ACK field in the segment.

Referenced by TcpFormatNetbuf(), and TcpInput().

TCP_SEQNO _TCP_SEG::End
The sequence of the last byte + 1, include SYN/FIN. End-Seq = SEG.LEN.

Referenced by TcpAdjustSndQue(), TcpDeliverData(), TcpFormatNetbuf(), TcpGetSegmentSndQue(), TcpInput(), TcpQueueData(), TcpSendAck(), TcpSendZeroProbe(), TcpToSendData(), TcpTransmitSegment(), TcpTrimSegment(), and TcpVerifySegment().

UINT8 _TCP_SEG::Flag
TCP header flags.

Referenced by TcpDeliverData(), TcpFormatNetbuf(), TcpGetSegmentSndQue(), TcpInput(), TcpSendAck(), TcpSendZeroProbe(), TcpToSendData(), TcpTransmitSegment(), TcpTrimSegment(), and TcpVerifySegment().

TCP_SEQNO _TCP_SEG::Seq
Starting sequence number.

Referenced by TcpAdjustSndQue(), TcpDeliverData(), TcpFormatNetbuf(), TcpGetSegmentSndQue(), TcpInput(), TcpQueueData(), TcpSendAck(), TcpSendZeroProbe(), TcpToSendData(), TcpTransmitSegment(), TcpTrimSegment(), and TcpVerifySegment().

UINT16 _TCP_SEG::Urg
Valid if URG flag is set.

Referenced by TcpFormatNetbuf(), TcpInput(), TcpTransmitSegment(), and TcpTrimSegment().

UINT32 _TCP_SEG::Wnd
TCP window size field.

Referenced by TcpFormatNetbuf(), and TcpInput().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_TCP_SERVICE_DATA Struct Reference

Data Fields

UINT32	Signature
EFI_HANDLE	ControllerHandle
EFI_HANDLE	DriverBindingHandle
UINT8	IpVersion
IP_IO *	IpIo
EFI_SERVICE_BINDING_PROTOCOL	ServiceBinding
LIST_ENTRY	SocketList

Field Documentation

EFI_HANDLE _TCP_SERVICE_DATA::ControllerHandle

Referenced by TcpCreateService(), TcpCreateSocketCallback(), and TcpInstallDevicePath().

EFI_HANDLE _TCP_SERVICE_DATA::DriverBindingHandle

Referenced by TcpCreateService(), TcpCreateSocketCallback(), TcpDestroySocketCallback(), and TcpServiceBindingCreateChild().

IP_IO* _TCP_SERVICE_DATA::IpIo

Referenced by Tcp6RefreshNeighbor(), TcpAttachPcb(), TcpCreateService(), TcpCreateSocketCallback(), TcpDestroyService(), TcpDestroySocketCallback(), TcpDetachPcb(), TcpDispatcher(), TcpGetRcvMss(), and TcpSendIpPacket().

UINT8 _TCP_SERVICE_DATA::IpVersion

Referenced by TcpCreateService(), and TcpServiceBindingCreateChild().

EFI_SERVICE_BINDING_PROTOCOL _TCP_SERVICE_DATA::ServiceBinding

Referenced by TcpCreateService().

UINT32 _TCP_SERVICE_DATA::Signature

Referenced by TcpCreateService().

LIST_ENTRY _TCP_SERVICE_DATA::SocketList

Referenced by TcpCreateService(), TcpCreateSocketCallback(), and TcpDestroyService().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

_TCP_SOCKET Struct Reference

Data Fields

UINT32	Signature Signature of the socket.
EFI_HANDLE	SockHandle The virtual handle of the socket.
EFI_HANDLE	DriverBinding Socket's driver binding protocol.
EFI_DEVICE_PATH_PROTOCOL *	ParentDevicePath
EFI_DEVICE_PATH_PROTOCOL *	DevicePath
LIST_ENTRY	Link
UINT8	ConfigureState
SOCK_TYPE	Type
UINT8	State
UINT16	Flag
EFI_LOCK	Lock The lock of socket.
SOCK_BUFFER	SndBuffer Send buffer of application's data.
SOCK_BUFFER	RcvBuffer Receive buffer of received data.
EFI_STATUS	SockError The error returned by low layer protocol.
BOOLEAN	InDestroy
UINT32	BackLog the limit of connection to this socket
UINT32	ConnCnt the current count of connections to it
SOCKET *	Parent listening parent that accept the connection
LIST_ENTRY	ConnectionList the connections maintained by this socket
LIST_ENTRY	ListenTokenList
LIST_ENTRY	RcvTokenList
LIST_ENTRY	SndTokenList
LIST_ENTRY	ProcessingSndTokenList
SOCK_COMPLETION_TOKEN *	ConnectionToken app's token to signal if connected
SOCK_COMPLETION_TOKEN *	CloseToken app's token to signal if closed
SOCK_PROTO_HANDLER	ProtoHandler The request handler of protocol.
UINT8	ProtoReserved [PROTO_RESERVED_LEN] Data fields reserved for protocol.
UINT8	IpVersion
NET_PROTOCOL	NetProtocol TCP or UDP protocol socket used.
SOCK_CREATE_CALLBACK	CreateCallback Callback after created.
SOCK_DESTROY_CALLBACK	DestroyCallback

Callback before destroied.

VOID * Context
The context of the callback.

Detailed Description

The socket structure representing a network service access point.

Field Documentation

UINT32 _TCP_SOCKET::BackLog
the limit of connection to this socket

Referenced by SockCreate().

SOCK_COMPLETION_TOKEN* _TCP_SOCKET::CloseToken
app's token to signal if closed

UINT8 _TCP_SOCKET::ConfigureState

Referenced by SockClone(), Tcp4Configure(), and Tcp6Configure().

UINT32 _TCP_SOCKET::ConnCnt
the current count of connections to it

Referenced by SockAccept(), SockCreate(), and SockWakeListenToken().

LIST_ENTRY _TCP_SOCKET::ConnectionList
the connections maintained by this socket

Referenced by SockCreate().

SOCK_COMPLETION_TOKEN* _TCP_SOCKET::ConnectionToken
app's token to signal if connected

VOID* _TCP_SOCKET::Context
The context of the callback.

Referenced by SockCreate().

SOCK_CREATE_CALLBACK _TCP_SOCKET::CreateCallback
Callback after created.

Referenced by SockCreate().

SOCK_DESTROY_CALLBACK _TCP_SOCKET::DestroyCallback
Callback before destroied.

Referenced by SockCreate().

EFI_DEVICE_PATH_PROTOCOL* _TCP_SOCKET::DevicePath

Referenced by TcpFlushPcb().

EFI_HANDLE _TCP_SOCKET::DriverBinding
Socket's driver binding protocol.

Referenced by SockCreate().

UINT16 _TCP_SOCKET::Flag

BOOLEAN _TCP_SOCKET::InDestroy

UINT8 _TCP_SOCKET::IpVersion

Referenced by SockCreate(), TcpDispatcher(), TcpFindTcbByPeer(), TcpIcmpInput(), TcpLocateListenTcb(), and TcpLocateTcb().

LIST_ENTRY _TCP_SOCKET::Link

Referenced by SockCreate().

LIST_ENTRY _TCP_SOCKET::ListenTokenList

Referenced by SockCreate(), and SockWakeListenToken().

EFI_LOCK _TCP_SOCKET::Lock
The lock of socket.

Referenced by SockCreate(), and SockCreateChild().

NET_PROTOCOL _TCP_SOCKET::NetProtocol
TCP or UDP protocol socket used.

Referenced by SockCreate().

SOCKET* _TCP_SOCKET::Parent
listening parent that accept the connection

Referenced by SockAccept(), SockCreate(), and SockWakeListenToken().

EFI_DEVICE_PATH_PROTOCOL* _TCP_SOCKET::ParentDevicePath

LIST_ENTRY _TCP_SOCKET::ProcessingSndTokenList

Referenced by SockCreate().

SOCK_PROTO_HANDLER _TCP_SOCKET::ProtoHandler
The request handler of protocol.

Referenced by SockCreate(), SockCreateChild(), Tcp4Poll(), and Tcp6Poll().

UINT8 _TCP_SOCKET::ProtoReserved[PROTO_RESERVED_LEN]
Data fields reserved for protocol.

Referenced by SockCreate(), Tcp6RefreshNeighbor(), TcpCloneTcb(), and TcpSendIpPacket().

SOCK_BUFFER _TCP_SOCKET::RcvBuffer
Receive buffer of received data.

Referenced by SockCreate().

LIST_ENTRY _TCP_SOCKET::RcvTokenList

Referenced by SockCreate().

UINT32 _TCP_SOCKET::Signature
Signature of the socket.

Referenced by SockCreate().

SOCK_BUFFER _TCP_SOCKET::SndBuffer
Send buffer of application's data.

Referenced by SockCreate().

LIST_ENTRY _TCP_SOCKET::SndTokenList

Referenced by SockCreate().

EFI_STATUS _TCP_SOCKET::SockError
The error returned by low layer protocol.

Referenced by SockCreate().

EFI_HANDLE _TCP_SOCKET::SockHandle
The virtual handle of the socket.

Referenced by SockAccept(), SockCreate(), TcpDestroyChildEntryInHandleBuffer(), TcpFlushPcb(), and TcpServiceBindingCreateChild().

UINT8 _TCP_SOCKET::State

Referenced by SockCreate(), and Tcp4GetMode().

SOCK_TYPE _TCP_SOCKET::Type

Referenced by SockCreate().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_UDP6_INSTANCE_DATA Struct Reference

Data Fields

UINT32	Signature
LIST_ENTRY	Link
UDP6_SERVICE_DATA *	Udp6Service
EFI_UDP6_PROTOCOL	Udp6Proto
EFI_UDP6_CONFIG_DATA	ConfigData
EFI_HANDLE	ChildHandle
BOOLEAN	Configured
BOOLEAN	IsNoMapping
NET_MAP	TxTokens
NET_MAP	RxTokens
NET_MAP	McastIps
LIST_ENTRY	RcvdDgramQue
LIST_ENTRY	DeliveredDgramQue
UINT16	HeadSum
EFI_STATUS	IcmpError
IP_IO_IP_INFO *	IpInfo
BOOLEAN	InDestroy

Field Documentation

EFI_HANDLE _UDP6_INSTANCE_DATA::ChildHandle

Referenced by Udp6DestroyChildEntryInHandleBuffer(), Udp6ServiceBindingCreateChild(), and Udp6ServiceBindingDestroyChild().

EFI_UDP6_CONFIG_DATA _UDP6_INSTANCE_DATA::ConfigData

Referenced by Udp6CheckTimeout(), Udp6Configure(), Udp6FindInstanceByPort(), Udp6GetModeData(), and Udp6Transmit().

BOOLEAN _UDP6_INSTANCE_DATA::Configured

Referenced by Udp6Cancel(), Udp6CheckTimeout(), Udp6Configure(), Udp6DeliverDgram(), Udp6EnqueueDgram(), Udp6FindInstanceByPort(), Udp6GetModeData(), Udp6Groups(), Udp6IcmpHandler(), Udp6Receive(), and Udp6Transmit().

LIST_ENTRY _UDP6_INSTANCE_DATA::DeliveredDgramQue

Referenced by Udp6Configure().

UINT16 _UDP6_INSTANCE_DATA::HeadSum

Referenced by Udp6Configure(), and Udp6Transmit().

EFI_STATUS _UDP6_INSTANCE_DATA::IcmpError

Referenced by Udp6IcmpHandler().

BOOLEAN _UDP6_INSTANCE_DATA::InDestroy

Referenced by Udp6ServiceBindingDestroyChild().

IP_IO_IP_INFO* _UDP6_INSTANCE_DATA::IpInfo

Referenced by Udp6Configure(), Udp6GetModeData(), Udp6Groups(), Udp6Poll(), Udp6ServiceBindingCreateChild(), Udp6ServiceBindingDestroyChild(), and Udp6Transmit().

BOOLEAN _UDP6_INSTANCE_DATA::IsNoMapping

Referenced by Udp6Configure().

LIST_ENTRY _UDP6_INSTANCE_DATA::Link

Referenced by Udp6ServiceBindingCreateChild(), and Udp6ServiceBindingDestroyChild().

NET_MAP _UDP6_INSTANCE_DATA::McastIps

Referenced by Udp6Groups().

LIST_ENTRY _UDP6_INSTANCE_DATA::RcvdDgramQue

Referenced by Udp6CheckTimeout(), and Udp6EnqueueDgram().

NET_MAP _UDP6_INSTANCE_DATA::RxTokens

Referenced by Udp6Receive(), and Udp6Transmit().

UINT32 _UDP6_INSTANCE_DATA::Signature

NET_MAP _UDP6_INSTANCE_DATA::TxTokens

Referenced by Udp6DgramSent(), Udp6Receive(), and Udp6Transmit().

EFI_UDP6_PROTOCOL _UDP6_INSTANCE_DATA::Udp6Proto

Referenced by Udp6Configure(), Udp6ServiceBindingCreateChild(), and Udp6ServiceBindingDestroyChild().

UDP6_SERVICE_DATA* _UDP6_INSTANCE_DATA::Udp6Service

Referenced by Udp6Configure(), and Udp6Transmit().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_UDP6_RXDATA_WRAP Struct Reference

Data Fields

LIST_ENTRY	Link
NET_BUF *	Packet
UINT32	TimeoutTick
EFI_UDP6_RECEIVE_DATA	RxData

Field Documentation

LIST_ENTRY _UDP6_RXDATA_WRAP::Link

Referenced by Udp6EnqueueDgram(), Udp6InstanceDeliverDgram(), Udp6RecycleRxDataWrap(), and Udp6WrapRxData().

NET_BUF* _UDP6_RXDATA_WRAP::Packet

Referenced by Udp6InstanceDeliverDgram(), Udp6RecycleRxDataWrap(), and Udp6WrapRxData().

EFI_UDP6_RECEIVE_DATA _UDP6_RXDATA_WRAP::RxData

Referenced by Udp6InstanceDeliverDgram(), Udp6RecycleRxDataWrap(), and Udp6WrapRxData().

UINT32 _UDP6_RXDATA_WRAP::TimeoutTick

Referenced by Udp6CheckTimeout(), and Udp6WrapRxData().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_UDP6_SERVICE_DATA Struct Reference

Data Fields

UINT32	Signature
EFI_SERVICE_BINDING_PROTOCOL	ServiceBinding
EFI_HANDLE	ImageHandle
EFI_HANDLE	ControllerHandle
LIST_ENTRY	ChildrenList
UINTN	ChildrenNumber
IP_IO *	IpIo
EFI_EVENT	TimeoutEvent

Field Documentation

LIST_ENTRY _UDP6_SERVICE_DATA::ChildrenList

Referenced by Udp6CheckTimeout(), Udp6Configure(), Udp6DriverBindingStop(), and Udp6ServiceBindingCreateChild().

UINTN _UDP6_SERVICE_DATA::ChildrenNumber

Referenced by Udp6ServiceBindingCreateChild(), and Udp6ServiceBindingDestroyChild().

EFI_HANDLE _UDP6_SERVICE_DATA::ControllerHandle

EFI_HANDLE _UDP6_SERVICE_DATA::ImageHandle

IP_IO* _UDP6_SERVICE_DATA::IpIo

Referenced by Udp6ServiceBindingCreateChild(), Udp6ServiceBindingDestroyChild(), and Udp6Transmit().

EFI_SERVICE_BINDING_PROTOCOL _UDP6_SERVICE_DATA::ServiceBinding

Referenced by Udp6DriverBindingStart(), and Udp6DriverBindingStop().

UINT32 _UDP6_SERVICE_DATA::Signature

EFI_EVENT _UDP6_SERVICE_DATA::TimeoutEvent

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

_VAR_CHECK_ITEM Struct Reference

Data Fields

CHAR16 * FlagStr
UINT32 FlagID
UINT32 ConflictMask
VAR_CHECK_FLAG_TYPE FlagType

Field Documentation

UINT32 _VAR_CHECK_ITEM::ConflictMask

UINT32 _VAR_CHECK_ITEM::FlagID

CHAR16* _VAR_CHECK_ITEM::FlagStr

VAR_CHECK_FLAG_TYPE _VAR_CHECK_ITEM::FlagType

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

DELETE_POLICY_ENTRY_CONTEXT Struct Reference

Data Fields

EFI_IPSEC_CONFIG_DATA_TYPE DataType
POLICY_ENTRY_INDEXER Indexer
EFI_STATUS Status

Field Documentation

EFI_IPSEC_CONFIG_DATA_TYPE DELETE_POLICY_ENTRY_CONTEXT::DataType

Referenced by FlushOrDeletePolicyEntry().

POLICY_ENTRY_INDEXER DELETE_POLICY_ENTRY_CONTEXT::Indexer

Referenced by FlushOrDeletePolicyEntry().

EFI_STATUS DELETE_POLICY_ENTRY_CONTEXT::Status

Referenced by FlushOrDeletePolicyEntry().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

DNS4_CACHE Struct Reference

Data Fields

LIST_ENTRY AllCacheLink
EFI_DNS4_CACHE_ENTRY DnsCache

Field Documentation

LIST_ENTRY DNS4_CACHE::AllCacheLink

Referenced by DnsOnTimerUpdate(), and UpdateDns4Cache().

EFI_DNS4_CACHE_ENTRY DNS4_CACHE::DnsCache

Referenced by Dns4GetModeData(), Dns4HostNameToIp(), DnsOnTimerUpdate(), DnsUnload(), and UpdateDns4Cache().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

DNS4_SERVER_INFOR Struct Reference

Data Fields

UINT32 * ServerCount
EFI_IPv4_ADDRESS * ServerList

Field Documentation

UINT32* DNS4_SERVER_INFOR::ServerCount

Referenced by GetDns4ServerFromDhcp4().

EFI_IPv4_ADDRESS* DNS4_SERVER_INFOR::ServerList

Referenced by GetDns4ServerFromDhcp4().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

DNS4_SERVER_IP Struct Reference

Data Fields

LIST_ENTRY AllServerLink
EFI_IPv4_ADDRESS Dns4ServerIp

Field Documentation

LIST_ENTRY DNS4_SERVER_IP::AllServerLink

Referenced by AddDns4ServerIp().

EFI_IPv4_ADDRESS DNS4_SERVER_IP::Dns4ServerIp

Referenced by AddDns4ServerIp(), and Dns4GetModeData().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

DNS4_TOKEN_ENTRY Struct Reference

Data Fields

UINT32	PacketToLive
CHAR16 *	QueryHostName
EFI_IPv4_ADDRESS	QueryIpAddress
EFI_DNS4_COMPLETION_TOKEN *	Token

Field Documentation

UINT32 DNS4_TOKEN_ENTRY::PacketToLive

Referenced by Dns4HostNameToIp(), DnsOnPacketSent(), and DnsOnTimerRetransmit().

CHAR16* DNS4_TOKEN_ENTRY::QueryHostName

Referenced by Dns4HostNameToIp(), and ParseDnsResponse().

EFI_IPv4_ADDRESS DNS4_TOKEN_ENTRY::QueryIpAddress

EFI_DNS4_COMPLETION_TOKEN* DNS4_TOKEN_ENTRY::Token

Referenced by Dns4CancelTokens(), Dns4HostNameToIp(), DnsOnPacketSent(), DnsOnTimerRetransmit(), and ParseDnsResponse().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

DNS6_CACHE Struct Reference

Data Fields

LIST_ENTRY AllCacheLink
EFI_DNS6_CACHE_ENTRY DnsCache

Field Documentation

LIST_ENTRY DNS6_CACHE::AllCacheLink

Referenced by DnsOnTimerUpdate(), and UpdateDns6Cache().

EFI_DNS6_CACHE_ENTRY DNS6_CACHE::DnsCache

Referenced by Dns6GetModeData(), Dns6HostNameToIp(), DnsOnTimerUpdate(), DnsUnload(), and UpdateDns6Cache().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

DNS6_SERVER_INFOR Struct Reference

Data Fields

UINT32 * ServerCount
EFI_IPv6_ADDRESS * ServerList

Field Documentation

UINT32* DNS6_SERVER_INFOR::ServerCount

Referenced by GetDns6ServerFromDhcp6(), and ParseDhcp6Ack().

EFI_IPv6_ADDRESS* DNS6_SERVER_INFOR::ServerList

Referenced by GetDns6ServerFromDhcp6(), and ParseDhcp6Ack().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

DNS6_SERVER_IP Struct Reference

Data Fields

LIST_ENTRY AllServerLink
EFI_IPv6_ADDRESS Dns6ServerIp

Field Documentation

LIST_ENTRY DNS6_SERVER_IP::AllServerLink

Referenced by AddDns6ServerIp().

EFI_IPv6_ADDRESS DNS6_SERVER_IP::Dns6ServerIp

Referenced by AddDns6ServerIp(), and Dns6GetModeData().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

DNS6_TOKEN_ENTRY Struct Reference

Data Fields

UINT32	PacketToLive
CHAR16 *	QueryHostName
EFI_IPv6_ADDRESS	QueryIpAddress
EFI_DNS6_COMPLETION_TOKEN *	Token

Field Documentation

UINT32 DNS6_TOKEN_ENTRY::PacketToLive

Referenced by Dns6HostNameToIp(), DnsOnPacketSent(), and DnsOnTimerRetransmit().

CHAR16* DNS6_TOKEN_ENTRY::QueryHostName

Referenced by Dns6HostNameToIp(), and ParseDnsResponse().

EFI_IPv6_ADDRESS DNS6_TOKEN_ENTRY::QueryIpAddress

EFI_DNS6_COMPLETION_TOKEN* DNS6_TOKEN_ENTRY::Token

Referenced by Dns6CancelTokens(), Dns6HostNameToIp(), DnsOnPacketSent(), DnsOnTimerRetransmit(), and ParseDnsResponse().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

DNS_ANSWER_SECTION Struct Reference

Data Fields

UINT16 Type
UINT16 Class
UINT32 Ttl
UINT16 DataLength

Field Documentation

UINT16 DNS_ANSWER_SECTION::Class

Referenced by ParseDnsResponse().

UINT16 DNS_ANSWER_SECTION::DataLength

Referenced by ParseDnsResponse().

UINT32 DNS_ANSWER_SECTION::Ttl

Referenced by ParseDnsResponse().

UINT16 DNS_ANSWER_SECTION::Type

Referenced by ParseDnsResponse().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT Struct Reference

Data Fields

EFI_SERVICE_BINDING_PROTOCOL * ServiceBinding
UINTN NumberOfChildren
EFI_HANDLE * ChildHandleBuffer

Field Documentation

EFI_HANDLE* DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer

Referenced by Dns4DriverBindingStop(), and Dns6DriverBindingStop().

UINTN DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren

Referenced by Dns4DriverBindingStop(), and Dns6DriverBindingStop().

EFI_SERVICE_BINDING_PROTOCOL*
DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding

Referenced by Dns4DriverBindingStop(), and Dns6DriverBindingStop().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

DNS_HEADER Struct Reference

Data Fields

UINT16 Identification
DNS_FLAGS Flags
UINT16 QuestionsNum
UINT16 AnswersNum
UINT16 AuthorityNum
UINT16 AditionalNum

Field Documentation

UINT16 DNS_HEADER::AditionalNum

Referenced by ConstructDNSQueryIp(), and ParseDnsResponse().

UINT16 DNS_HEADER::AnswersNum

Referenced by ConstructDNSQueryIp(), and ParseDnsResponse().

UINT16 DNS_HEADER::AuthorityNum

Referenced by ConstructDNSQueryIp(), and ParseDnsResponse().

DNS_FLAGS DNS_HEADER::Flags

Referenced by ConstructDNSQueryIp(), and ParseDnsResponse().

UINT16 DNS_HEADER::Identification

Referenced by ConstructDNSQueryIp(), IsValidDnsResponse(), and ParseDnsResponse().

UINT16 DNS_HEADER::QuestionsNum

Referenced by ConstructDNSQueryIp(), and ParseDnsResponse().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Data Fields

DNS_QUERY_SECTION Struct Reference

Data Fields

UINT16 Type
UINT16 Class

Field Documentation

UINT16 DNS_QUERY_SECTION::Class

Referenced by ConstructDNSQueryIp(), and ParseDnsResponse().

UINT16 DNS_QUERY_SECTION::Type

Referenced by ConstructDNSQueryIp(), IsValidDnsResponse(), and ParseDnsResponse().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

EDIT_POLICY_ENTRY_CONTEXT Struct Reference

Data Fields

EFI_IPSEC_CONFIG_DATA_TYPE	DataType
EFI_IPSEC_CONFIG_SELECTOR *	Selector
VOID *	Data
UINT32	Mask
POLICY_ENTRY_INDEXER	Indexer
EFI_STATUS	Status

Field Documentation

VOID* EDIT_POLICY_ENTRY_CONTEXT::Data

Referenced by EditPolicyEntry().

EFI_IPSEC_CONFIG_DATA_TYPE EDIT_POLICY_ENTRY_CONTEXT::DataType

Referenced by EditPolicyEntry().

POLICY_ENTRY_INDEXER EDIT_POLICY_ENTRY_CONTEXT::Indexer

Referenced by EditPolicyEntry().

UINT32 EDIT_POLICY_ENTRY_CONTEXT::Mask

Referenced by EditPolicyEntry().

EFI_IPSEC_CONFIG_SELECTOR* EDIT_POLICY_ENTRY_CONTEXT::Selector

Referenced by EditPolicyEntry().

EFI_STATUS EDIT_POLICY_ENTRY_CONTEXT::Status

Referenced by EditPolicyEntry().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Data Fields

HASH_DATA_FRAGMENT Struct Reference

Data Fields

UINTN DataSize
UINT8 * Data

Field Documentation

UINT8* HASH_DATA_FRAGMENT::Data

Referenced by Ikev2DecryptPacket(), Ikev2EncryptPacket(), Ikev2GenerateCertificatePayload(), Ikev2GenerateSaKeys(), Ikev2PskGenerateAuthPayload(), IpSecEspAuthVerifyPayload(), and IpSecEspOutboundPacket().

UINTN HASH_DATA_FRAGMENT::DataSize

Referenced by Ikev2DecryptPacket(), Ikev2EncryptPacket(), Ikev2GenerateCertificatePayload(), Ikev2GenerateSaKeys(), Ikev2PskGenerateAuthPayload(), IpSecEspAuthVerifyPayload(), and IpSecEspOutboundPacket().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

HII_VENDOR_DEVICE_PATH Struct Reference

Data Fields

VENDOR_DEVICE_PATH VendorDevicePath
EFI_DEVICE_PATH_PROTOCOL End

Detailed Description

HII specific Vendor Device Path definition.

Field Documentation

EFI_DEVICE_PATH_PROTOCOL HII_VENDOR_DEVICE_PATH::End

VENDOR_DEVICE_PATH HII_VENDOR_DEVICE_PATH::VendorDevicePath

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

HTTP4_IO_CONFIG_DATA Struct Reference

Data Fields

EFI_HTTP_VERSION	HttpVersion
UINT32	RequestTimeOut
UINT32	ResponseTimeOut
BOOLEAN	UseDefaultAddress
EFI_IPv4_ADDRESS	LocalIp
EFI_IPv4_ADDRESS	SubnetMask
UINT16	LocalPort

Field Documentation

EFI_HTTP_VERSION HTTP4_IO_CONFIG_DATA::HttpVersion

Referenced by HttpBootCreateHttpIo().

EFI_IPv4_ADDRESS HTTP4_IO_CONFIG_DATA::LocalIp

Referenced by HttpBootCreateHttpIo().

UINT16 HTTP4_IO_CONFIG_DATA::LocalPort

UINT32 HTTP4_IO_CONFIG_DATA::RequestTimeOut

Referenced by HttpBootCreateHttpIo().

UINT32 HTTP4_IO_CONFIG_DATA::ResponseTimeOut

EFI_IPv4_ADDRESS HTTP4_IO_CONFIG_DATA::SubnetMask

Referenced by HttpBootCreateHttpIo().

BOOLEAN HTTP4_IO_CONFIG_DATA::UseDefaultAddress

- Data Fields

HTTP_BOOT_CACHE_CONTENT Struct Reference

Data Fields

LIST_ENTRY	Link
EFI_HTTP_REQUEST_DATA *	RequestData
HTTP_IO_RESOPNSE_DATA *	ResponseData
UINTN	EntityLength
LIST_ENTRY	EntityDataList

Field Documentation

LIST_ENTRY HTTP_BOOT_CACHE_CONTENT::EntityDataList

Referenced by HttpBootGetBootFile(), HttpBootGetBootFileCallback(), and HttpBootGetFileFromCache().

UINTN HTTP_BOOT_CACHE_CONTENT::EntityLength

Referenced by HttpBootGetBootFile(), and HttpBootGetFileFromCache().

LIST_ENTRY HTTP_BOOT_CACHE_CONTENT::Link

Referenced by HttpBootFreeCacheList(), and HttpBootGetBootFile().

EFI_HTTP_REQUEST_DATA* HTTP_BOOT_CACHE_CONTENT::RequestData

Referenced by HttpBootGetBootFile(), and HttpBootGetFileFromCache().

HTTP_IO_RESOPNSE_DATA* HTTP_BOOT_CACHE_CONTENT::ResponseData

Referenced by HttpBootGetBootFile().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

HTTP_BOOT_CALLBACK_DATA Struct Reference

Data Fields

EFI_STATUS	Status
HTTP_BOOT_CACHE_CONTENT *	Cache
BOOLEAN	NewBlock
UINT8 *	Block
UINTN	CopyedSize
UINTN	BufferSize
UINT8 *	Buffer

Field Documentation

UINT8* HTTP_BOOT_CALLBACK_DATA::Block

Referenced by HttpBootGetBootFile(), and HttpBootGetBootFileCallback().

UINT8* HTTP_BOOT_CALLBACK_DATA::Buffer

Referenced by HttpBootGetBootFile(), and HttpBootGetBootFileCallback().

UINTN HTTP_BOOT_CALLBACK_DATA::BufferSize

Referenced by HttpBootGetBootFile(), and HttpBootGetBootFileCallback().

HTTP_BOOT_CACHE_CONTENT* HTTP_BOOT_CALLBACK_DATA::Cache

Referenced by HttpBootGetBootFile(), and HttpBootGetBootFileCallback().

UINTN HTTP_BOOT_CALLBACK_DATA::CopyedSize

Referenced by HttpBootGetBootFile(), and HttpBootGetBootFileCallback().

BOOLEAN HTTP_BOOT_CALLBACK_DATA::NewBlock

Referenced by HttpBootGetBootFile(), and HttpBootGetBootFileCallback().

EFI_STATUS HTTP_BOOT_CALLBACK_DATA::Status

- Data Structures
- Files

- Data Structures
- Data Fields

HTTP_BOOT_DHCP4_OPTION_ARCH Struct Reference

Data Fields

UINT16 Type

Field Documentation

UINT16 HTTP_BOOT_DHCP4_OPTION_ARCH::Type

Referenced by HttpBootBuildDhcp4Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

HTTP_BOOT_DHCP4_OPTION_CLID Struct Reference

Data Fields

UINT8 ClassIdentifier [11]
UINT8 ArchitecturePrefix [5]
UINT8 ArchitectureType [5]
UINT8 Lit3 [1]
UINT8 InterfaceName [4]
UINT8 Lit4 [1]
UINT8 UndiMajor [3]
UINT8 UndiMinor [3]

Field Documentation

UINT8 HTTP_BOOT_DHCP4_OPTION_CLID::ArchitecturePrefix[5]

UINT8 HTTP_BOOT_DHCP4_OPTION_CLID::ArchitectureType[5]

Referenced by HttpBootBuildDhcp4Options().

UINT8 HTTP_BOOT_DHCP4_OPTION_CLID::ClassIdentifier[11]

UINT8 HTTP_BOOT_DHCP4_OPTION_CLID::InterfaceName[4]

Referenced by HttpBootBuildDhcp4Options().

UINT8 HTTP_BOOT_DHCP4_OPTION_CLID::Lit3[1]

UINT8 HTTP_BOOT_DHCP4_OPTION_CLID::Lit4[1]

UINT8 HTTP_BOOT_DHCP4_OPTION_CLID::UndiMajor[3]

Referenced by HttpBootBuildDhcp4Options().

UINT8 HTTP_BOOT_DHCP4_OPTION_CLID::UndiMinor[3]

Referenced by HttpBootBuildDhcp4Options().

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

HTTP_BOOT_DHCP4_OPTION_ENTRY Union Reference

Data Fields

HTTP_BOOT_DHCP4_OPTION_PARA * Para
HTTP_BOOT_DHCP4_OPTION_UNDI * Undi
HTTP_BOOT_DHCP4_OPTION_ARCH * Arch
HTTP_BOOT_DHCP4_OPTION_CLID * Clid
HTTP_BOOT_DHCP4_OPTION_UUID * Uuid
HTTP_BOOT_DHCP4_OPTION_MESG * Mesg
HTTP_BOOT_DHCP4_OPTION_MAX_MESG_SIZE * MaxMesgSize

Field Documentation

HTTP_BOOT_DHCP4_OPTION_ARCH* HTTP_BOOT_DHCP4_OPTION_ENTRY::Arch

Referenced by HttpBootBuildDhcp4Options().

HTTP_BOOT_DHCP4_OPTION_CLID* HTTP_BOOT_DHCP4_OPTION_ENTRY::Clid

Referenced by HttpBootBuildDhcp4Options().

HTTP_BOOT_DHCP4_OPTION_MAX_MESG_SIZE*
HTTP_BOOT_DHCP4_OPTION_ENTRY::MaxMesgSize

HTTP_BOOT_DHCP4_OPTION_MESG* HTTP_BOOT_DHCP4_OPTION_ENTRY::Mesg

HTTP_BOOT_DHCP4_OPTION_PARA* HTTP_BOOT_DHCP4_OPTION_ENTRY::Para

Referenced by HttpBootBuildDhcp4Options().

HTTP_BOOT_DHCP4_OPTION_UNDI* HTTP_BOOT_DHCP4_OPTION_ENTRY::Undi

Referenced by HttpBootBuildDhcp4Options().

HTTP_BOOT_DHCP4_OPTION_UUID* HTTP_BOOT_DHCP4_OPTION_ENTRY::Uuid

Referenced by HttpBootBuildDhcp4Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

HTTP_BOOT_DHCP4_OPTION_MAX_MESG_SIZE Struct Reference

Data Fields

UINT16 Size

Field Documentation

UINT16 HTTP_BOOT_DHCP4_OPTION_MAX_MESG_SIZE::Size

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

HTTP_BOOT_DHCP4_OPTION_MESG Struct Reference

Data Fields

UINT8 Type

Field Documentation

UINT8 HTTP_BOOT_DHCP4_OPTION_MESG::Type

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

HTTP_BOOT_DHCP4_OPTION_PARA Struct Reference

Data Fields

UINT8 ParaList [135]

Field Documentation

UINT8 HTTP_BOOT_DHCP4_OPTION_PARA::ParaList[135]

Referenced by HttpBootBuildDhcp4Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

HTTP_BOOT_DHCP4_OPTION_UNDI Struct Reference

Data Fields

UINT8 Type
UINT8 MajorVer
UINT8 MinorVer

Field Documentation

UINT8 HTTP_BOOT_DHCP4_OPTION_UNDI::MajorVer

Referenced by HttpBootBuildDhcp4Options().

UINT8 HTTP_BOOT_DHCP4_OPTION_UNDI::MinorVer

Referenced by HttpBootBuildDhcp4Options().

UINT8 HTTP_BOOT_DHCP4_OPTION_UNDI::Type

Referenced by HttpBootBuildDhcp4Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

HTTP_BOOT_DHCP4_OPTION_UUID Struct Reference

Data Fields

UINT8 Type
UINT8 Guid [16]

Field Documentation

UINT8 HTTP_BOOT_DHCP4_OPTION_UUID::Guid[16]

Referenced by HttpBootBuildDhcp4Options().

UINT8 HTTP_BOOT_DHCP4_OPTION_UUID::Type

Referenced by HttpBootBuildDhcp4Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

HTTP_BOOT_DHCP4_PACKET Union Reference

Data Fields

EFI_DHCP4_PACKET Offer
EFI_DHCP4_PACKET Ack
UINT8 Buffer [HTTP_BOOT_DHCP4_PACKET_MAX_SIZE]

Field Documentation

EFI_DHCP4_PACKET HTTP_BOOT_DHCP4_PACKET::Ack

UINT8 HTTP_BOOT_DHCP4_PACKET::Buffer[HTTP_BOOT_DHCP4_PACKET_MAX_SIZE]

EFI_DHCP4_PACKET HTTP_BOOT_DHCP4_PACKET::Offer

Referenced by HttpBootCacheDhcp4Offer(), and HttpBootDhcp4CallBack().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

HTTP_BOOT_DHCP4_PACKET_CACHE Struct Reference

Data Fields

HTTP_BOOT_DHCP4_PACKET Packet
HTTP_BOOT_OFFER_TYPE OfferType
VOID * UriParser
EFI_DHCP4_PACKET_OPTION * OptList [HTTP_BOOT_DHCP4_TAG_INDEX_MAX]

Field Documentation

HTTP_BOOT_OFFER_TYPE HTTP_BOOT_DHCP4_PACKET_CACHE::OfferType

Referenced by HttpBootCacheDhcp4Offer(), and HttpBootExtractUriInfo().

EFI_DHCP4_PACKET_OPTION*
HTTP_BOOT_DHCP4_PACKET_CACHE::OptList[HTTP_BOOT_DHCP4_TAG_INDEX_MAX]

Referenced by HttpBootExtractUriInfo().

HTTP_BOOT_DHCP4_PACKET HTTP_BOOT_DHCP4_PACKET_CACHE::Packet

Referenced by HttpBootCacheDhcp4Offer(), and HttpBootDhcp4CallBack().

VOID* HTTP_BOOT_DHCP4_PACKET_CACHE::UriParser

Referenced by HttpBootExtractUriInfo().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

HTTP_BOOT_DHCP_PACKET_CACHE Union Reference

Data Fields

HTTP_BOOT_DHCP4_PACKET_CACHE Dhcp4

Field Documentation

HTTP_BOOT_DHCP4_PACKET_CACHE HTTP_BOOT_DHCP_PACKET_CACHE::Dhcp4

Referenced by HttpBootDhcp4CallBack().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

HTTP_BOOT_ENTITY_DATA Struct Reference

Data Fields

LIST_ENTRY Link
UINT8 * Block
UINT8 * DataStart
UINTN DataLength

Field Documentation

UINT8* HTTP_BOOT_ENTITY_DATA::Block

Referenced by HttpBootFreeCache(), and HttpBootGetBootFileCallback().

UINTN HTTP_BOOT_ENTITY_DATA::DataLength

Referenced by HttpBootGetBootFileCallback(), and HttpBootGetFileFromCache().

UINT8* HTTP_BOOT_ENTITY_DATA::DataStart

Referenced by HttpBootGetBootFileCallback(), and HttpBootGetFileFromCache().

LIST_ENTRY HTTP_BOOT_ENTITY_DATA::Link

Referenced by HttpBootFreeCache(), and HttpBootGetBootFileCallback().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

HTTP_BOOT_OPTION_BOOT_ITEM Struct Reference

Data Fields

UINT16 Type
UINT16 Layer

Field Documentation

UINT16 HTTP_BOOT_OPTION_BOOT_ITEM::Layer

UINT16 HTTP_BOOT_OPTION_BOOT_ITEM::Type

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

HTTP_BOOT_URI_CONTENT Struct Reference

Data Fields

CHAR8 *	Scheme
CHAR8 *	Authority
CHAR8 *	Path
CHAR8 *	Query
CHAR8 *	Fragment
CHAR8 *	RegName TODO: may not required in HTTP URL.
BOOLEAN	AddrIsOk Point to somewhere in Authority.
EFI_IP_ADDRESS	Address
UINT16	Port

Field Documentation

EFI_IP_ADDRESS HTTP_BOOT_URI_CONTENT::Address

BOOLEAN HTTP_BOOT_URI_CONTENT::AddrIsOk
Point to somewhere in Authority.

CHAR8* HTTP_BOOT_URI_CONTENT::Authority

CHAR8* HTTP_BOOT_URI_CONTENT::Fragment

CHAR8* HTTP_BOOT_URI_CONTENT::Path

UINT16 HTTP_BOOT_URI_CONTENT::Port

CHAR8* HTTP_BOOT_URI_CONTENT::Query

CHAR8* HTTP_BOOT_URI_CONTENT::RegName
TODO: may not required in HTTP URL.

CHAR8* HTTP_BOOT_URI_CONTENT::Scheme

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page

- Data Structures
- Files

- Data Structures
- Data Fields

HTTP_BOOT_VENDOR_OPTION Struct Reference

Data Fields

UINT32 Reserved

Field Documentation

UINT32 HTTP_BOOT_VENDOR_OPTION::Reserved

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

HTTP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT Struct Reference

Data Fields

EFI_SERVICE_BINDING_PROTOCOL * ServiceBinding
UINTN NumberOfChildren
EFI_HANDLE * ChildHandleBuffer

Field Documentation

EFI_HANDLE* HTTP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer

Referenced by HttpDxeDriverBindingStop().

UINTN HTTP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren

Referenced by HttpDxeDriverBindingStop().

EFI_SERVICE_BINDING_PROTOCOL*
HTTP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding

Referenced by HttpDxeDriverBindingStop().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

HTTP_IO Struct Reference

Data Fields

UINT8	IpVersion
EFI_HANDLE	Image
EFI_HANDLE	Controller
EFI_HANDLE	Handle
EFI_HTTP_PROTOCOL *	Http
EFI_HTTP_TOKEN	ReqToken
EFI_HTTP_MESSAGE	ReqMessage
EFI_HTTP_TOKEN	RspToken
EFI_HTTP_MESSAGE	RspMessage
BOOLEAN	IsTxDone
BOOLEAN	IsRxDone

Field Documentation

EFI_HANDLE HTTP_IO::Controller

EFI_HANDLE HTTP_IO::Handle

EFI_HTTP_PROTOCOL* HTTP_IO::Http

EFI_HANDLE HTTP_IO::Image

UINT8 HTTP_IO::IpVersion

BOOLEAN HTTP_IO::IsRxDone

BOOLEAN HTTP_IO::IsTxDone

EFI_HTTP_MESSAGE HTTP_IO::ReqMessage

EFI_HTTP_TOKEN HTTP_IO::ReqToken

EFI_HTTP_MESSAGE HTTP_IO::RspMessage

EFI_HTTP_TOKEN HTTP_IO::RspToken

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

HTTP_IO_CONFIG_DATA Union Reference

Data Fields

HTTP4_IO_CONFIG_DATA Config4

Field Documentation

HTTP4_IO_CONFIG_DATA HTTP_IO_CONFIG_DATA::Config4

Referenced by HttpBootCreateHttpIo().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

HTTP_IO_HEADER Struct Reference

Data Fields

UINTN	MaxHeaderCount
UINTN	HeaderCount
EFI_HTTP_HEADER *	Headers

Field Documentation

UINTN HTTP_IO_HEADER::HeaderCount

Referenced by HttpBootGetBootFile().

EFI_HTTP_HEADER* HTTP_IO_HEADER::Headers

Referenced by HttpBootCreateHeader(), and HttpBootGetBootFile().

UINTN HTTP_IO_HEADER::MaxHeaderCount

Referenced by HttpBootCreateHeader().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

HTTP_IO_RESOPNSE_DATA Struct Reference

Data Fields

EFI_HTTP_RESPONSE_DATA Response
UINTN HeaderCount
EFI_HTTP_HEADER * Headers
UINTN BodyLength
CHAR8 * Body

Field Documentation

CHAR8* HTTP_IO_RESOPNSE_DATA::Body

Referenced by HttpBootGetBootFile().

UINTN HTTP_IO_RESOPNSE_DATA::BodyLength

Referenced by HttpBootGetBootFile().

UINTN HTTP_IO_RESOPNSE_DATA::HeaderCount

Referenced by HttpBootGetBootFile().

EFI_HTTP_HEADER* HTTP_IO_RESOPNSE_DATA::Headers

Referenced by HttpBootGetBootFile().

EFI_HTTP_RESPONSE_DATA HTTP_IO_RESOPNSE_DATA::Response

Referenced by HttpBootGetBootFile().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

HTTP_TCP_TOKEN_WRAP Struct Reference

Data Fields

EFI_TCP4_IO_TOKEN	TxToken
EFI_TCP4_TRANSMIT_DATA	TxData
BOOLEAN	IsTxDone
EFI_TCP4_IO_TOKEN	RxToken
EFI_TCP4_RECEIVE_DATA	RxData
BOOLEAN	IsRxDone
UINTN	BodyLen
EFI_HTTP_METHOD	Method

Field Documentation

UINTN HTTP_TCP_TOKEN_WRAP::BodyLen

BOOLEAN HTTP_TCP_TOKEN_WRAP::IsRxDone

Referenced by HttpTcpReceiveNotifyDpc().

BOOLEAN HTTP_TCP_TOKEN_WRAP::IsTxDone

Referenced by HttpTcpNotReady(), HttpTcpTransmit(), and HttpTcpTransmitNotifyDpc().

EFI_HTTP_METHOD HTTP_TCP_TOKEN_WRAP::Method

Referenced by EfiHttpRequest().

EFI_TCP4_RECEIVE_DATA HTTP_TCP_TOKEN_WRAP::RxData

Referenced by HttpCreateTcp4RxEvent(), and HttpTcpReceiveNotifyDpc().

EFI_TCP4_IO_TOKEN HTTP_TCP_TOKEN_WRAP::RxToken

Referenced by EfiHttpResponse(), HttpCancelTokens(), HttpCreateTcp4RxEvent(), and HttpTcpReceiveNotifyDpc().

EFI_TCP4_TRANSMIT_DATA HTTP_TCP_TOKEN_WRAP::TxData

Referenced by HttpCreateTcp4TxEvent().

EFI_TCP4_IO_TOKEN HTTP_TCP_TOKEN_WRAP::TxToken

Referenced by EfiHttpRequest(), HttpCancelTokens(), HttpCreateTcp4TxEvent(), and HttpTcpTransmitNotifyDpc().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

HTTP_TOKEN_WRAP Struct Reference

Data Fields

EFI_HTTP_TOKEN * HttpToken
HTTP_PROTOCOL * HttpInstance
HTTP_TCP_TOKEN_WRAP TcpWrap

Field Documentation

HTTP_PROTOCOL* HTTP_TOKEN_WRAP::HttpInstance

Referenced by EfiHttpRequest(), EfiHttpResponse(), HttpBodyParserCallback(), HttpTcpReceiveNotifyDpc(), HttpTcpTransmit(), and HttpTcpTransmitNotifyDpc().

EFI_HTTP_TOKEN* HTTP_TOKEN_WRAP::HttpToken

Referenced by EfiHttpRequest(), EfiHttpResponse(), HttpResponseWorker(), HttpTcpReceiveNotifyDpc(), HttpTcpTransmit(), and HttpTcpTransmitNotifyDpc().

HTTP_TCP_TOKEN_WRAP HTTP_TOKEN_WRAP::TcpWrap

Referenced by EfiHttpRequest(), EfiHttpResponse(), HttpCancelTokens(), HttpTcpNotReady(), HttpTcpReceiveNotifyDpc(), HttpTcpTransmit(), and HttpTcpTransmitNotifyDpc().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKE_ALG_GUID_INFO Struct Reference

Data Fields

UINT8 AlgorithmId
EFI_GUID * AlgGuid
UINT8 AlgSize
UINT8 BlockSize
UINTN KeyMateLen

Field Documentation

EFI_GUID* IKE_ALG_GUID_INFO::AlgGuid

UINT8 IKE_ALG_GUID_INFO::AlgorithmId

UINT8 IKE_ALG_GUID_INFO::AlgSize

UINT8 IKE_ALG_GUID_INFO::BlockSize

UINTN IKE_ALG_GUID_INFO::KeyMateLen

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IKE_HEADER Struct Reference

Data Fields

UINT64 InitiatorCookie
UINT64 ResponderCookie
UINT8 NextPayload
UINT8 Version
UINT8 ExchangeType
UINT8 Flags
UINT32 MessageId
UINT32 Length

Field Documentation

UINT8 IKE_HEADER::ExchangeType

Referenced by IkeDispatch(), Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), Ikev2InfoGenerator(), and Ikev2InitPskGenerator().

UINT8 IKE_HEADER::Flags

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), Ikev2InfoGenerator(), and Ikev2InitPskGenerator().

UINT64 IKE_HEADER::InitiatorCookie

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), Ikev2InfoGenerator(), and Ikev2InitPskGenerator().

UINT32 IKE_HEADER::Length

Referenced by IkePacketFromNetbuf().

UINT32 IKE_HEADER::MessageId

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), Ikev2InfoGenerator(), and Ikev2InitPskGenerator().

UINT8 IKE_HEADER::NextPayload

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), Ikev2InfoGenerator(), and Ikev2InitPskGenerator().

UINT64 IKE_HEADER::ResponderCookie

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), Ikev2InfoGenerator(), and Ikev2InitPskGenerator().

UINT8 IKE_HEADER::Version

Referenced by IkeDispatch(), Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), Ikev2InfoGenerator(), and Ikev2InitPskGenerator().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IKE_PACKET Struct Reference

Data Fields

UINTN	RefCount
BOOLEAN	IsHdrExt
IKE_HEADER *	Header
BOOLEAN	IsPayloadsBufExt
UINT8 *	PayloadsBuf
UINTN	PayloadTotalSize
LIST_ENTRY	PayloadList
EFI_IP_ADDRESS	RemotePeerIp
BOOLEAN	IsEncoded
UINT32	Spi
BOOLEAN	IsDeleteInfo
IPSEC_PRIVATE_DATA *	Private

Field Documentation

IKE_HEADER* IKE_PACKET::Header

Referenced by IkeDispatch(), IkePacketAlloc(), IkePacketFromNetbuf(), Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), Ikev2InfoGenerator(), and Ikev2InitPskGenerator().

BOOLEAN IKE_PACKET::IsDeleteInfo

Referenced by Ikev2InfoGenerator(), and Ikev2OnPacketSent().

BOOLEAN IKE_PACKET::IsEncoded

BOOLEAN IKE_PACKET::IsHdrExt

BOOLEAN IKE_PACKET::IsPayloadsBufExt

LIST_ENTRY IKE_PACKET::PayloadList

Referenced by IkePacketAlloc().

UINT8* IKE_PACKET::PayloadsBuf

Referenced by IkePacketFromNetbuf().

UINTN IKE_PACKET::PayloadTotalSize

Referenced by IkePacketFromNetbuf().

IPSEC_PRIVATE_DATA* IKE_PACKET::Private

Referenced by Ikev2InfoGenerator(), and Ikev2OnPacketSent().

UINTN IKE_PACKET::RefCount

Referenced by IkePacketAlloc().

EFI_IP_ADDRESS IKE_PACKET::RemotePeerIp

Referenced by IkeDispatch(), and Ikev2OnPacketSent().

UINT32 IKE_PACKET::Spi

Referenced by Ikev2InfoGenerator(), and Ikev2OnPacketSent().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IKE_PAYLOAD Struct Reference

Data Fields

UINT32 Signature
BOOLEAN IsPayloadBufExt
UINT8 PayloadType
UINT8 * PayloadBuf
UINTN PayloadSize
LIST_ENTRY ByPacket

Field Documentation

LIST_ENTRY IKE_PAYLOAD::ByPacket

BOOLEAN IKE_PAYLOAD::IsPayloadBufExt

Referenced by Ikev2DecodePacket().

UINT8* IKE_PAYLOAD::PayloadBuf

Referenced by IkeNetbufFromPacket(), Ikev2AuthCertParser(), Ikev2AuthPskParser(), Ikev2CertGenerateAuthPayload(), Ikev2DecodePacket(), Ikev2EncodePacket(), Ikev2EncryptPacket(), Ikev2GenerateCertIdPayload(), Ikev2GenerateCertificatePayload(), Ikev2GenerateCpPayload(), Ikev2GenerateDeletePayload(), Ikev2GenerateIdPayload(), Ikev2GenerateKePayload(), Ikev2GenerateNoncePayload(), Ikev2GenerateNotifyPayload(), Ikev2GenerateSaPayload(), Ikev2GenerateTsPayload(), Ikev2InfoParser(), Ikev2InitCertGenerator(), Ikev2InitPskParser(), and Ikev2PskGenerateAuthPayload().

UINTN IKE_PAYLOAD::PayloadSize

Referenced by IkeNetbufFromPacket(), Ikev2AuthCertParser(), Ikev2AuthPskParser(), Ikev2CertGenerateAuthPayload(), Ikev2DecodePacket(), Ikev2EncodePacket(), Ikev2EncryptPacket(), Ikev2GenerateCertIdPayload(), Ikev2GenerateCertificatePayload(), Ikev2GenerateCpPayload(), Ikev2GenerateDeletePayload(), Ikev2GenerateIdPayload(), Ikev2GenerateKePayload(), Ikev2GenerateNoncePayload(), Ikev2GenerateNotifyPayload(), Ikev2GenerateTsPayload(), Ikev2InitPskParser(), and Ikev2PskGenerateAuthPayload().

UINT8 IKE_PAYLOAD::PayloadType

Referenced by Ikev2AuthCertParser(), Ikev2AuthPskParser(), Ikev2CertGenerateAuthPayload(), Ikev2ChildExchangeRequestType(), Ikev2DecodePacket(), Ikev2EncodePacket(), Ikev2EncryptPacket(), Ikev2GenerateCertIdPayload(), Ikev2GenerateCertificatePayload(), Ikev2GenerateCpPayload(), Ikev2GenerateDeletePayload(), Ikev2GenerateIdPayload(), Ikev2GenerateKePayload(), Ikev2GenerateNoncePayload(), Ikev2GenerateNotifyPayload(), Ikev2GenerateSaPayload(),

Ikev2GenerateTsPayload(), Ikev2InfoParser(), Ikev2InitPskParser(), and Ikev2PskGenerateAuthPayload().

UINT32 IKE_PAYLOAD::Signature

Referenced by IkePayloadAlloc().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IKE_SA_ATTR_UNION Union Reference

Data Fields

UINT16 AttrLength
UINT16 AttrValue

Field Documentation

UINT16 IKE_SA_ATTR_UNION::AttrLength

Referenced by Ikev2InitializeSaData().

UINT16 IKE_SA_ATTR_UNION::AttrValue

Referenced by Ikev2DecodeSa(), Ikev2EncodeSa(), and Ikev2ParseProposalData().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKE_SA_ATTRIBUTE Struct Reference

Data Fields

UINT16 AttrType
IKE_SA_ATTR_UNION Attr

Field Documentation

IKE_SA_ATTR_UNION IKE_SA_ATTRIBUTE::Attr

Referenced by Ikev2DecodeSa(), Ikev2EncodeSa(), Ikev2InitializeSaData(), and Ikev2ParseProposalData().

UINT16 IKE_SA_ATTRIBUTE::AttrType

Referenced by Ikev2DecodeSa(), Ikev2EncodeSa(), Ikev2InitializeSaData(), and Ikev2ParseProposalData().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IKE_UDP_SERVICE Struct Reference

Data Fields

UINT32	Signature
UINT8	IpVersion
LIST_ENTRY	List
LIST_ENTRY *	ListHead
EFI_HANDLE	NicHandle
EFI_HANDLE	ImageHandle
UDP_IO *	Input
UDP_IO *	Output
EFI_IP_ADDRESS	DefaultAddress
BOOLEAN	IsConfigured

Field Documentation

EFI_IP_ADDRESS IKE_UDP_SERVICE::DefaultAddress

Referenced by IkeOpenInputUdp4(), and IkeOpenInputUdp6().

EFI_HANDLE IKE_UDP_SERVICE::ImageHandle

Referenced by IkeOpenInputUdp4(), and IkeOpenInputUdp6().

UDP_IO* IKE_UDP_SERVICE::Input

Referenced by IkeDispatch(), IkeOpenInputUdp4(), IkeOpenInputUdp6(), and IpSecStop().

UINT8 IKE_UDP_SERVICE::IpVersion

Referenced by IkeDispatch(), IkeOpenInputUdp4(), IkeOpenInputUdp6(), Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), and Ikev2MatchSpdEntry().

BOOLEAN IKE_UDP_SERVICE::IsConfigured

Referenced by IkeOpenInputUdp4(), and IkeOpenInputUdp6().

LIST_ENTRY IKE_UDP_SERVICE::List

Referenced by IkeOpenInputUdp4(), IkeOpenInputUdp6(), and IpSecStop().

LIST_ENTRY* IKE_UDP_SERVICE::ListHead

Referenced by IkeDispatch(), IkeOpenInputUdp4(), and IkeOpenInputUdp6().

EFI_HANDLE IKE_UDP_SERVICE::NicHandle

Referenced by IkeLookupUdp(), IkeOpenInputUdp4(), and IkeOpenInputUdp6().

UDP_IO* IKE_UDP_SERVICE::Output

Referenced by Ikev2NegotiateInfo(), and IpSecStop().

UINT32 IKE_UDP_SERVICE::Signature

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IKEV2_AUTH Struct Reference

Data Fields

IKEV2_COMMON_PAYLOAD_HEADER	Header
UINT8	AuthMethod
UINT8	Reserved1
UINT16	Reserved2

Field Documentation

UINT8 IKEV2_AUTH::AuthMethod

Referenced by Ikev2CertGenerateAuthPayload(), and Ikev2PskGenerateAuthPayload().

IKEV2_COMMON_PAYLOAD_HEADER IKEV2_AUTH::Header

Referenced by Ikev2CertGenerateAuthPayload(), and Ikev2PskGenerateAuthPayload().

UINT8 IKEV2_AUTH::Reserved1

UINT16 IKEV2_AUTH::Reserved2

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_CERT Struct Reference

Data Fields

IKEV2_COMMON_PAYLOAD_HEADER Header
UINT8 CertEncoding

Field Documentation

UINT8 IKEV2_CERT::CertEncoding

Referenced by Ikev2GenerateCertificatePayload().

IKEV2_COMMON_PAYLOAD_HEADER IKEV2_CERT::Header

Referenced by Ikev2GenerateCertificatePayload().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_CERT_REQ Struct Reference

Data Fields

IKEV2_COMMON_PAYLOAD_HEADER Header
UINT8 CertEncoding

Field Documentation

UINT8 IKEV2_CERT_REQ::CertEncoding

IKEV2_COMMON_PAYLOAD_HEADER IKEV2_CERT_REQ::Header

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_CFG Struct Reference

Data Fields

IKEV2_COMMON_PAYLOAD_HEADER	Header
UINT8	CfgType
UINT8	Reserve1
UINT16	Reserve2

Field Documentation

UINT8 IKEV2_CFG::CfgType

Referenced by Ikev2GenerateCpPayload().

IKEV2_COMMON_PAYLOAD_HEADER IKEV2_CFG::Header

Referenced by Ikev2GenerateCpPayload().

UINT8 IKEV2_CFG::Reserve1

UINT16 IKEV2_CFG::Reserve2

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_CFG_ATTRIBUTES Struct Reference

Data Fields

UINT16 AttritType
UINT16 ValueLength

Field Documentation

UINT16 IKEV2_CFG_ATTRIBUTES::AttritType

Referenced by Ikev2DecodePayload(), Ikev2EncodePayload(), and Ikev2GenerateCpPayload().

UINT16 IKEV2_CFG_ATTRIBUTES::ValueLength

Referenced by Ikev2DecodePayload(), Ikev2EncodePayload(), and Ikev2GenerateCpPayload().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_CHILD_SA_SESSION Struct Reference

Data Fields

UINT32	Signature
IKEV2_SESSION_COMMON	SessionCommon
IKEV2_SA_SESSION *	IkeSaSession
UINT32	MessageId
IKEV2_SA_DATA *	SaData
UINT8	IpsecProtocol
UINT32	LocalPeerSpi
UINT32	RemotePeerSpi
UINT8 *	NiBlock
UINTN	NiBlkSize
UINT8 *	NrBlock
UINTN	NrBlkSize
SA_KEYMATS	ChildKeymats
IKEV2_DH_BUFFER *	DhBuffer
IPSEC_SPD_ENTRY *	Spd
EFI_IPSEC_SPD_SELECTOR *	SpdSelector
UINT16	ProtoId
UINT16	RemotePort
UINT16	LocalPort
LIST_ENTRY	ByIkeSa
LIST_ENTRY	ByDelete

Field Documentation

LIST_ENTRY IKEV2_CHILD_SA_SESSION::ByDelete

Referenced by Ikev2InfoParser(), and Ikev2LifetimeNotify().

LIST_ENTRY IKEV2_CHILD_SA_SESSION::ByIkeSa

Referenced by Ikev2HandleSa().

SA_KEYMATS IKEV2_CHILD_SA_SESSION::ChildKeymats

IKEV2_DH_BUFFER* IKEV2_CHILD_SA_SESSION::DhBuffer

IKEV2_SA_SESSION* IKEV2_CHILD_SA_SESSION::IkeSaSession

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), Ikev2ChildSaSessionAlloc(), Ikev2CreateChildGenerator(), Ikev2DecryptPacket(), Ikev2EncryptPacket(), Ikev2InfoGenerator(), Ikev2LifetimeNotify(), Ikev2ResendNotify(), and

Ikev2SaSessionFree().

UINT8 IKEV2_CHILD_SA_SESSION::IpsecProtocol

UINT32 IKEV2_CHILD_SA_SESSION::LocalPeerSpi

Referenced by Ikev2ChildSaSessionAlloc(), Ikev2ChildSaSessionLookupBySpi(), Ikev2ChildSaSessionRemove(), Ikev2ChildSaSilentDelete(), Ikev2HandleSa(), Ikev2InfoGenerator(), Ikev2InitializeSaData(), Ikev2LifetimeNotify(), Ikev2OnPacketSent(), Ikev2ResendNotify(), and Ikev2SaSessionFree().

UINT16 IKEV2_CHILD_SA_SESSION::LocalPort

Referenced by Ikev2AuthCertParser(), Ikev2AuthPskParser(), Ikev2ChildSaSessionCreate(), Ikev2MatchSpdEntry(), and Ikev2NegotiateChildSa().

UINT32 IKEV2_CHILD_SA_SESSION::MessageId

Referenced by Ikev2ChildSaSessionAlloc(), Ikev2ChildSaSessionLookupByMid(), Ikev2CreateChildGenerator(), Ikev2InfoGenerator(), and Ikev2InfoParser().

UINTN IKEV2_CHILD_SA_SESSION::NiBlkSize

Referenced by Ikev2ChildSaSessionCreate().

UINT8* IKEV2_CHILD_SA_SESSION::NiBlock

Referenced by Ikev2ChildSaSessionCreate().

UINTN IKEV2_CHILD_SA_SESSION::NrBlkSize

Referenced by Ikev2ChildSaSessionCreate().

UINT8* IKEV2_CHILD_SA_SESSION::NrBlock

Referenced by Ikev2ChildSaSessionCreate().

UINT16 IKEV2_CHILD_SA_SESSION::ProtoId

Referenced by Ikev2AuthCertParser(), Ikev2AuthPskParser(), Ikev2ChildSaSessionCreate(), Ikev2MatchSpdEntry(), and Ikev2NegotiateChildSa().

UINT32 IKEV2_CHILD_SA_SESSION::RemotePeerSpi

Referenced by Ikev2ChildSaSessionLookupBySpi(), Ikev2ChildSaSessionRemove(), Ikev2ChildSaSilentDelete(), and Ikev2InfoParser().

UINT16 IKEV2_CHILD_SA_SESSION::RemotePort

Referenced by Ikev2AuthCertParser(), Ikev2AuthPskParser(), Ikev2ChildSaSessionCreate(), Ikev2MatchSpdEntry(), and Ikev2NegotiateChildSa().

IKEV2_SA_DATA* IKEV2_CHILD_SA_SESSION::SaData

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2HandleSa(), and Ikev2NegotiateChildSa().

IKEV2_SESSION_COMMON IKEV2_CHILD_SA_SESSION::SessionCommon

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), Ikev2ChildSaSessionAlloc(), Ikev2ChildSaSessionCreate(), Ikev2CreateChildGenerator(), Ikev2HandleSa(), Ikev2InfoGenerator(), Ikev2InfoParser(), Ikev2MatchSpdEntry(), Ikev2NegotiateChildSa(), Ikev2NegotiateInfo(), and Ikev2ResendNotify().

UINT32 IKEV2_CHILD_SA_SESSION::Signature

Referenced by Ikev2ChildSaSessionAlloc().

IPSEC_SPD_ENTRY* IKEV2_CHILD_SA_SESSION::Spd

Referenced by Ikev2AuthCertParser(), Ikev2AuthPskParser(), Ikev2ChildSaSessionCreate(), Ikev2MatchSpdEntry(), and Ikev2NegotiateChildSa().

EFI_IPSEC_SPD_SELECTOR* IKEV2_CHILD_SA_SESSION::SpdSelector

Referenced by Ikev2AuthCertParser(), and Ikev2AuthPskParser().

IKEV2_COMMON_PAYLOAD_HEADER Struct Reference

Data Fields

UINT8 NextPayload
UINT8 Reserved
UINT16 PayloadLength

Field Documentation

UINT8 IKEV2_COMMON_PAYLOAD_HEADER::NextPayload

Referenced by Ikev2CertGenerateAuthPayload(), Ikev2DecodePacket(), Ikev2DecodeSa(), Ikev2EncodeSa(), Ikev2GenerateCertIdPayload(), Ikev2GenerateCertificatePayload(), Ikev2GenerateCpPayload(), Ikev2GenerateDeletePayload(), Ikev2GenerateIdPayload(), Ikev2GenerateKePayload(), Ikev2GenerateNoncePayload(), Ikev2GenerateNotifyPayload(), Ikev2GenerateSaPayload(), Ikev2GenerateTsPayload(), and Ikev2PskGenerateAuthPayload().

UINT16 IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength

Referenced by Ikev2CertGenerateAuthPayload(), Ikev2DecodePacket(), Ikev2DecodePayload(), Ikev2DecodeSa(), Ikev2EncodePayload(), Ikev2EncodeSa(), Ikev2GenerateCertIdPayload(), Ikev2GenerateCertificatePayload(), Ikev2GenerateCpPayload(), Ikev2GenerateDeletePayload(), Ikev2GenerateIdPayload(), Ikev2GenerateKePayload(), Ikev2GenerateNoncePayload(), Ikev2GenerateNotifyPayload(), Ikev2GenerateTsPayload(), Ikev2ParserNotifyCookiePayload(), and Ikev2PskGenerateAuthPayload().

UINT8 IKEV2_COMMON_PAYLOAD_HEADER::Reserved

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_DELETE Struct Reference

Data Fields

IKEV2_COMMON_PAYLOAD_HEADER	Header
UINT8	ProtocolId
UINT8	SpiSize
UINT16	NumSpis

Field Documentation

IKEV2_COMMON_PAYLOAD_HEADER IKEV2_DELETE::Header

Referenced by Ikev2GenerateDeletePayload().

UINT16 IKEV2_DELETE::NumSpis

Referenced by Ikev2DecodePayload(), Ikev2EncodePayload(), Ikev2GenerateDeletePayload(), and Ikev2InfoParser().

UINT8 IKEV2_DELETE::ProtocolId

Referenced by Ikev2GenerateDeletePayload().

UINT8 IKEV2_DELETE::SpiSize

Referenced by Ikev2GenerateDeletePayload(), and Ikev2InfoParser().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IKEV2_DH_BUFFER Struct Reference

Data Fields

UINT8 * GxBuffer
UINTN GxSize
UINT8 * GyBuffer
UINTN GySize
UINT8 * GxyBuffer
UINTN GxySize
UINT8 * DhContext

Field Documentation

UINT8* IKEV2_DH_BUFFER::DhContext

Referenced by Ikev2DhBufferFree(), and Ikev2GenerateSaDhPublicKey().

UINT8* IKEV2_DH_BUFFER::GxBuffer

Referenced by Ikev2DhBufferFree(), Ikev2GenerateKePayload(), and Ikev2GenerateSaDhPublicKey().

UINTN IKEV2_DH_BUFFER::GxSize

Referenced by Ikev2GenerateKePayload(), and Ikev2GenerateSaDhPublicKey().

UINT8* IKEV2_DH_BUFFER::GxyBuffer

Referenced by Ikev2DhBufferFree().

UINTN IKEV2_DH_BUFFER::GxySize

UINT8* IKEV2_DH_BUFFER::GyBuffer

Referenced by Ikev2DhBufferFree().

UINTN IKEV2_DH_BUFFER::GySize

- Data Structures
- Data Fields

IKEV2_ENCRYPTED Struct Reference

Data Fields

IKEV2_COMMON_PAYLOAD_HEADER Header

Field Documentation

IKEV2_COMMON_PAYLOAD_HEADER IKEV2_ENCRYPTED::Header

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IKEV2_ID Struct Reference

Data Fields

IKEV2_COMMON_PAYLOAD_HEADER	Header
UINT8	IdType
UINT8	Reserver1
UINT16	Reserver2

Field Documentation

IKEV2_COMMON_PAYLOAD_HEADER IKEV2_ID::Header

Referenced by Ikev2GenerateCertIdPayload(), and Ikev2GenerateIdPayload().

UINT8 IKEV2_ID::IdType

Referenced by Ikev2GenerateCertIdPayload(), and Ikev2GenerateIdPayload().

UINT8 IKEV2_ID::Reserver1

UINT16 IKEV2_ID::Reserver2

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IKEV2_INFO_EXCHANGE_CONTEXT Struct Reference

Data Fields

UINT32 MessageId
IKEV2_INFO_TYPE InfoType

Field Documentation

IKEV2_INFO_TYPE IKEV2_INFO_EXCHANGE_CONTEXT::InfoType

Referenced by Ikev2InfoGenerator(), and Ikev2InfoParser().

UINT32 IKEV2_INFO_EXCHANGE_CONTEXT::MessageId

Referenced by Ikev2InfoGenerator(), and Ikev2InfoParser().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_KEY_EXCHANGE Struct Reference

Data Fields

IKEV2_COMMON_PAYLOAD_HEADER Header
UINT16 DhGroup
UINT16 Reserved

Field Documentation

UINT16 IKEV2_KEY_EXCHANGE::DhGroup

Referenced by Ikev2DecodePayload(), Ikev2EncodePayload(), and Ikev2GenerateKePayload().

IKEV2_COMMON_PAYLOAD_HEADER IKEV2_KEY_EXCHANGE::Header

Referenced by Ikev2GenerateKePayload().

UINT16 IKEV2_KEY_EXCHANGE::Reserved

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_NONCE Struct Reference

Data Fields

IKEV2_COMMON_PAYLOAD_HEADER Header

Field Documentation

IKEV2_COMMON_PAYLOAD_HEADER IKEV2_NONCE::Header

Referenced by Ikev2GenerateNoncePayload().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IKEV2_NOTIFY Struct Reference

Data Fields

IKEV2_COMMON_PAYLOAD_HEADER Header
UINT8 ProtocolId
UINT8 SpiSize
UINT16 MessageType

Field Documentation

IKEV2_COMMON_PAYLOAD_HEADER IKEV2_NOTIFY::Header

Referenced by Ikev2GenerateNotifyPayload(), and Ikev2ParserNotifyCookiePayload().

UINT16 IKEV2_NOTIFY::MessageType

Referenced by Ikev2DecodePayload(), Ikev2EncodePayload(), Ikev2GenerateNotifyPayload(), and Ikev2ParserNotifyCookiePayload().

UINT8 IKEV2_NOTIFY::ProtocolId

Referenced by Ikev2GenerateNotifyPayload(), and Ikev2ParserNotifyCookiePayload().

UINT8 IKEV2_NOTIFY::SpiSize

Referenced by Ikev2GenerateNotifyPayload(), and Ikev2ParserNotifyCookiePayload().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_PACKET_HANDLER Struct Reference

Data Fields

IKEV2_PACKET_PARSER Parser
IKEV2_PACKET_GENERATOR Generator

Field Documentation

IKEV2_PACKET_GENERATOR IKEV2_PACKET_HANDLER::Generator

Referenced by Ikev2HandleChildSa(), Ikev2HandleSa(), Ikev2NegotiateChildSa(), Ikev2NegotiateInfo(), and Ikev2NegotiateSa().

IKEV2_PACKET_PARSER IKEV2_PACKET_HANDLER::Parser

Referenced by Ikev2HandleChildSa(), Ikev2HandleInfo(), and Ikev2HandleSa().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_PAD_LEN Struct Reference

Data Fields

UINT8 PadLength

Field Documentation

UINT8 IKEV2_PAD_LEN::PadLength

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IKEV2_PROPOSAL Struct Reference

Data Fields

IKEV2_COMMON_PAYLOAD_HEADER Header
UINT8 ProposalIndex
UINT8 ProtocolId
UINT8 SpiSize
UINT8 NumTransforms

Field Documentation

IKEV2_COMMON_PAYLOAD_HEADER IKEV2_PROPOSAL::Header

Referenced by Ikev2DecodeSa(), and Ikev2EncodeSa().

UINT8 IKEV2_PROPOSAL::NumTransforms

Referenced by Ikev2DecodeSa(), and Ikev2EncodeSa().

UINT8 IKEV2_PROPOSAL::ProposalIndex

Referenced by Ikev2DecodeSa(), and Ikev2EncodeSa().

UINT8 IKEV2_PROPOSAL::ProtocolId

Referenced by Ikev2DecodeSa(), and Ikev2EncodeSa().

UINT8 IKEV2_PROPOSAL::SpiSize

Referenced by Ikev2DecodeSa(), and Ikev2EncodeSa().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_PROPOSAL_DATA Struct Reference

Data Fields

UINT8 ProposalIndex
UINT8 ProtocolId
UINT8 * Spi
UINT8 NumTransforms

Field Documentation

UINT8 IKEV2_PROPOSAL_DATA::NumTransforms

Referenced by Ikev2ChildSaParseSaPayload(), Ikev2DecodeSa(), Ikev2EncodeSa(), Ikev2InitializeSaData(), and Ikev2SaParseSaPayload().

UINT8 IKEV2_PROPOSAL_DATA::ProposalIndex

Referenced by Ikev2DecodeSa(), Ikev2EncodeSa(), and Ikev2InitializeSaData().

UINT8 IKEV2_PROPOSAL_DATA::ProtocolId

Referenced by Ikev2ChildSaParseSaPayload(), Ikev2DecodeSa(), Ikev2EncodeSa(), Ikev2InitializeSaData(), and Ikev2SaParseSaPayload().

UINT8* IKEV2_PROPOSAL_DATA::Spi

Referenced by Ikev2ChildSaParseSaPayload(), Ikev2DecodeSa(), Ikev2EncodeSa(), and Ikev2InitializeSaData().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_SA Struct Reference

Data Fields

IKEV2_COMMON_PAYLOAD_HEADER Header

Field Documentation

IKEV2_COMMON_PAYLOAD_HEADER IKEV2_SA::Header

Referenced by Ikev2EncodeSa(), and Ikev2GenerateSaPayload().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_SA_DATA Struct Reference

Data Fields

IKEV2_SA SaHeader
UINTN NumProposals

Field Documentation

UINTN IKEV2_SA_DATA::NumProposals

Referenced by Ikev2DecodeSa(), and Ikev2InitializeSaData().

IKEV2_SA IKEV2_SA_DATA::SaHeader

Referenced by Ikev2GenerateSaPayload().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_SA_PARAMS Struct Reference

Data Fields

UINT16 LifeType
UINT64 LifeDuration
UINT16 EncAlgId
UINTN EnckeyLen
UINT16 Prf
UINT16 IntegAlgId
UINTN IntegKeyLen
UINT16 DhGroup
UINT8 ExtSeq

Field Documentation

UINT16 IKEV2_SA_PARAMS::DhGroup

UINT16 IKEV2_SA_PARAMS::EncAlgId

Referenced by Ikev2DecryptPacket(), Ikev2EncryptPacket(), Ikev2GenerateChildSaKeys(), and Ikev2GenerateSaKeys().

UINTN IKEV2_SA_PARAMS::EnckeyLen

UINT8 IKEV2_SA_PARAMS::ExtSeq

UINT16 IKEV2_SA_PARAMS::IntegAlgId

Referenced by Ikev2DecryptPacket(), Ikev2EncryptPacket(), Ikev2GenerateChildSaKeys(), and Ikev2GenerateSaKeys().

UINTN IKEV2_SA_PARAMS::IntegKeyLen

UINT64 IKEV2_SA_PARAMS::LifeDuration

UINT16 IKEV2_SA_PARAMS::LifeType

UINT16 IKEV2_SA_PARAMS::Prf

Referenced by Ikev2GenerateSaKeys().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IKEV2_SA_SESSION Struct Reference

Data Fields

UINT32	Signature
IKEV2_SESSION_COMMON	SessionCommon
UINT64	InitiatorCookie
UINT64	ResponderCookie
IKEV2_SA_DATA *	SaData
IKEV2_SESSION_KEYS *	IkeKeys
UINT8 *	NiBlock
UINTN	NiBlkSize
UINT8 *	NrBlock
UINTN	NrBlkSize
UINT8 *	NCookie
UINTN	NCookieSize
IPSEC_PAD_ENTRY *	Pad
IPSEC_SPD_ENTRY *	Spd
LIST_ENTRY	ChildSaSessionList
LIST_ENTRY	ChildSaEstablishSessionList
LIST_ENTRY	InfoMIDList
LIST_ENTRY	DeleteSaList
UINT8 *	InitPacket
UINTN	InitPacketSize
UINT8 *	RespPacket
UINTN	RespPacketSize
UINT32	MessageId
LIST_ENTRY	BySessionTable

Field Documentation

LIST_ENTRY IKEV2_SA_SESSION::BySessionTable

Referenced by Ikev2InfoParser(), Ikev2OnPacketSent(), and IpSecStop().

LIST_ENTRY IKEV2_SA_SESSION::ChildSaEstablishSessionList

Referenced by Ikev2ChildSaSessionReg(), Ikev2HandleSa(), Ikev2InfoParser(), Ikev2OnPacketSent(), Ikev2ResendNotify(), and Ikev2SaSessionAlloc().

LIST_ENTRY IKEV2_SA_SESSION::ChildSaSessionList

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), Ikev2HandleSa(), Ikev2NegotiateChildSa(), Ikev2ResendNotify(), and Ikev2SaSessionAlloc().

LIST_ENTRY IKEV2_SA_SESSION::DeleteSaList

Referenced by Ikev2InfoParser(), Ikev2LifetimeNotify(), Ikev2NegotiateInfo(), Ikev2OnPacketSent(), Ikev2ResendNotify(), and Ikev2SaSessionAlloc().

IKEV2_SESSION_KEYS* IKEV2_SA_SESSION::IkeKeys

Referenced by Ikev2DecryptPacket(), and Ikev2EncryptPacket().

LIST_ENTRY IKEV2_SA_SESSION::InfoMIDList

Referenced by Ikev2SaSessionAlloc().

UINT64 IKEV2_SA_SESSION::InitiatorCookie

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), Ikev2InfoGenerator(), Ikev2InitPskGenerator(), Ikev2InitPskParser(), Ikev2LifetimeNotify(), Ikev2OnPacketSent(), and Ikev2SaSessionAlloc().

UINT8* IKEV2_SA_SESSION::InitPacket

Referenced by Ikev2DecodePacket(), and Ikev2EncodePacket().

UINTN IKEV2_SA_SESSION::InitPacketSize

Referenced by Ikev2DecodePacket(), and Ikev2EncodePacket().

UINT32 IKEV2_SA_SESSION::MessageId

Referenced by Ikev2CreateChildGenerator(), Ikev2HandleChildSa(), Ikev2InfoGenerator(), and Ikev2SaSessionAlloc().

UINT8* IKEV2_SA_SESSION::NCookie

Referenced by Ikev2InitPskGenerator(), and Ikev2SaSessionAlloc().

UINTN IKEV2_SA_SESSION::NCookieSize

Referenced by Ikev2InitPskGenerator().

UINTN IKEV2_SA_SESSION::NiBlkSize

Referenced by Ikev2InitPskGenerator(), and Ikev2InitPskParser().

UINT8* IKEV2_SA_SESSION::NiBlock

Referenced by Ikev2InitPskGenerator(), and Ikev2InitPskParser().

UINTN IKEV2_SA_SESSION::NrBlkSize

Referenced by Ikev2InitPskGenerator(), and Ikev2InitPskParser().

UINT8* IKEV2_SA_SESSION::NrBlock

Referenced by Ikev2InitPskGenerator(), and Ikev2InitPskParser().

IPSEC_PAD_ENTRY* IKEV2_SA_SESSION::Pad

Referenced by Ikev2AuthCertGenerator(), Ikev2HandleSa(), and Ikev2NegotiateSa().

UINT64 IKEV2_SA_SESSION::ResponderCookie

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), Ikev2InfoGenerator(), Ikev2InitPskGenerator(), Ikev2InitPskParser(), Ikev2LifetimeNotify(), Ikev2OnPacketSent(), and Ikev2SaSessionAlloc().

UINT8* IKEV2_SA_SESSION::RespPacket

Referenced by Ikev2DecodePacket(), and Ikev2EncodePacket().

UINTN IKEV2_SA_SESSION::RespPacketSize

Referenced by Ikev2DecodePacket(), and Ikev2EncodePacket().

IKEV2_SA_DATA* IKEV2_SA_SESSION::SaData

Referenced by Ikev2InitPskGenerator(), and Ikev2NegotiateSa().

IKEV2_SESSION_COMMON IKEV2_SA_SESSION::SessionCommon

Referenced by IkeDeleteAllSas(), Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), Ikev2ChildSaSessionAlloc(), Ikev2CreateChildGenerator(), Ikev2DecryptPacket(), Ikev2EncryptPacket(), Ikev2HandleChildSa(), Ikev2HandleInfo(), Ikev2HandleSa(), Ikev2InfoGenerator(), Ikev2InfoParser(), Ikev2InitPskGenerator(), Ikev2InitPskParser(), Ikev2LifetimeNotify(), Ikev2NegotiateChildSa(), Ikev2NegotiateInfo(), Ikev2NegotiateSa(), Ikev2ResendNotify(), Ikev2SaSessionAlloc(), Ikev2SaSessionLookup(), and Ikev2SaSessionRemove().

UINT32 IKEV2_SA_SESSION::Signature

Referenced by Ikev2SaSessionAlloc().

IPSEC_SPD_ENTRY* IKEV2_SA_SESSION::Spd

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), and Ikev2NegotiateSa().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IKEV2_SESSION_COMMON Struct Reference

Data Fields

UINT8	IkeVer
IKE_SESSION_TYPE	IkeSessionType
BOOLEAN	IsInitiator
BOOLEAN	IsOnDeleting
IKEV2_SESSION_STATE	State
EFI_EVENT	TimeoutEvent
UINT64	TimeoutInterval
UINTN	RetryCount
IKE_PACKET *	LastSentPacket
IKEV2_SA_PARAMS *	SaParams
UINT16	PreferDhGroup
EFI_IP_ADDRESS	RemotePeerIp
EFI_IP_ADDRESS	LocalPeerIp
IKE_ON_PAYLOAD_FROM_NET	BeforeDecodePayload
IKE_ON_PAYLOAD_FROM_NET	AfterEncodePayload
IKE_UDP_SERVICE *	UdpService
IPSEC_PRIVATE_DATA *	Private

Field Documentation

IKE_ON_PAYLOAD_FROM_NET IKEV2_SESSION_COMMON::AfterEncodePayload

Referenced by Ikev2ChildSaSessionAlloc(), and Ikev2SaSessionAlloc().

IKE_ON_PAYLOAD_FROM_NET IKEV2_SESSION_COMMON::BeforeDecodePayload

Referenced by Ikev2ChildSaSessionAlloc(), and Ikev2SaSessionAlloc().

IKE_SESSION_TYPE IKEV2_SESSION_COMMON::IkeSessionType

Referenced by Ikev2ChildSaSessionAlloc(), Ikev2CreateChildGenerator(), Ikev2InfoGenerator(), Ikev2LifetimeNotify(), Ikev2ResendNotify(), and Ikev2SaSessionAlloc().

UINT8 IKEV2_SESSION_COMMON::IkeVer

Referenced by Ikev2ChildSaSessionAlloc(), and Ikev2SaSessionAlloc().

BOOLEAN IKEV2_SESSION_COMMON::IsInitiator

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), Ikev2ChildSaSessionCreate(), Ikev2CreateChildGenerator(), Ikev2HandleSa(),

Ikev2InfoGenerator(), Ikev2InfoParser(), Ikev2InitPskGenerator(), Ikev2InitPskParser(), Ikev2NegotiateChildSa(), and Ikev2NegotiateSa().

BOOLEAN IKEV2_SESSION_COMMON::IsOnDeleting

Referenced by Ikev2InfoGenerator().

IKE_PACKET* IKEV2_SESSION_COMMON::LastSentPacket

Referenced by Ikev2ResendNotify(), and Ikev2SendIkePacket().

EFI_IP_ADDRESS IKEV2_SESSION_COMMON::LocalPeerIp

Referenced by Ikev2ChildSaSessionAlloc(), Ikev2HandleSa(), Ikev2MatchSpdEntry(), Ikev2NegotiateSa(), and Ikev2SendIkePacket().

UINT16 IKEV2_SESSION_COMMON::PreferDhGroup

Referenced by Ikev2InitPskParser(), and Ikev2NegotiateSa().

IPSEC_PRIVATE_DATA* IKEV2_SESSION_COMMON::Private

Referenced by Ikev2ChildSaSessionAlloc(), Ikev2InfoGenerator(), Ikev2InfoParser(), Ikev2ResendNotify(), Ikev2SaSessionAlloc(), and Ikev2StoreSaData().

EFI_IP_ADDRESS IKEV2_SESSION_COMMON::RemotePeerIp

Referenced by Ikev2ChildSaSessionAlloc(), Ikev2HandleSa(), Ikev2MatchSpdEntry(), Ikev2NegotiateSa(), Ikev2ResendNotify(), Ikev2SaSessionLookup(), Ikev2SaSessionReg(), Ikev2SaSessionRemove(), and Ikev2SendIkePacket().

UINTN IKEV2_SESSION_COMMON::RetryCount

Referenced by Ikev2ResendNotify().

IKEV2_SA_PARAMS* IKEV2_SESSION_COMMON::SaParams

Referenced by Ikev2DecryptPacket(), and Ikev2EncryptPacket().

IKEV2_SESSION_STATE IKEV2_SESSION_COMMON::State

Referenced by IkeDeleteAllSas(), Ikev2AuthCertParser(), Ikev2AuthPskParser(), Ikev2ChildSaSessionCreate(), Ikev2HandleSa(), Ikev2InfoGenerator(), Ikev2InfoParser(), Ikev2InitPskParser(), Ikev2LifetimeNotify(), Ikev2NegotiateChildSa(), Ikev2NegotiateInfo(),

Ikev2NegotiateSa(), and Ikev2ResendNotify().

EFI_EVENT IKEV2_SESSION_COMMON::TimeoutEvent

Referenced by Ikev2ChildSaSessionAlloc(), Ikev2ChildSaSessionReg(), Ikev2NegotiateChildSa(), Ikev2SaSessionAlloc(), Ikev2SaSessionReg(), and Ikev2SendIkePacket().

UINT64 IKEV2_SESSION_COMMON::TimeoutInterval

Referenced by Ikev2SendIkePacket().

IKE_UDP_SERVICE* IKEV2_SESSION_COMMON::UdpService

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), Ikev2ChildSaSessionAlloc(), Ikev2MatchSpdEntry(), Ikev2NegotiateChildSa(), Ikev2NegotiateInfo(), Ikev2ResendNotify(), and Ikev2SaSessionAlloc().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_SESSION_KEYS Struct Reference

Data Fields

IKEV2_DH_BUFFER * DhBuffer
UINT8 * SkdKey
UINTN SkdKeySize
UINT8 * SkAiKey
UINTN SkAiKeySize
UINT8 * SkArKey
UINTN SkArKeySize
UINT8 * SkEiKey
UINTN SkEiKeySize
UINT8 * SkErKey
UINTN SkErKeySize
UINT8 * SkPiKey
UINTN SkPiKeySize
UINT8 * SkPrKey
UINTN SkPrKeySize

Field Documentation

IKEV2_DH_BUFFER* IKEV2_SESSION_KEYS::DhBuffer

Referenced by Ikev2GenerateKePayload(), Ikev2GenerateSaDhPublicKey(), and Ikev2SaSessionFree().

UINT8* IKEV2_SESSION_KEYS::SkAiKey

Referenced by Ikev2DecryptPacket(), Ikev2EncryptPacket(), and Ikev2SaSessionFree().

UINTN IKEV2_SESSION_KEYS::SkAiKeySize

Referenced by Ikev2DecryptPacket(), and Ikev2EncryptPacket().

UINT8* IKEV2_SESSION_KEYS::SkArKey

Referenced by Ikev2DecryptPacket(), Ikev2EncryptPacket(), and Ikev2SaSessionFree().

UINTN IKEV2_SESSION_KEYS::SkArKeySize

Referenced by Ikev2DecryptPacket(), and Ikev2EncryptPacket().

UINT8* IKEV2_SESSION_KEYS::SkdKey

Referenced by Ikev2SaSessionFree().

UINTN IKEV2_SESSION_KEYS::SkdKeySize

UINT8* IKEV2_SESSION_KEYS::SkEiKey

Referenced by Ikev2DecryptPacket(), Ikev2EncryptPacket(), and Ikev2SaSessionFree().

UINTN IKEV2_SESSION_KEYS::SkEiKeySize

Referenced by Ikev2DecryptPacket(), and Ikev2EncryptPacket().

UINT8* IKEV2_SESSION_KEYS::SkErKey

Referenced by Ikev2DecryptPacket(), Ikev2EncryptPacket(), and Ikev2SaSessionFree().

UINTN IKEV2_SESSION_KEYS::SkErKeySize

Referenced by Ikev2DecryptPacket(), and Ikev2EncryptPacket().

UINT8* IKEV2_SESSION_KEYS::SkPiKey

Referenced by Ikev2SaSessionFree().

UINTN IKEV2_SESSION_KEYS::SkPiKeySize

UINT8* IKEV2_SESSION_KEYS::SkPrKey

Referenced by Ikev2SaSessionFree().

UINTN IKEV2_SESSION_KEYS::SkPrKeySize

IKEV2_TRANSFORM Struct Reference

Data Fields

IKEV2_COMMON_PAYLOAD_HEADER Header
UINT8 TransformType
UINT8 Reserved
UINT16 TransformId

Field Documentation

IKEV2_COMMON_PAYLOAD_HEADER IKEV2_TRANSFORM::Header

Referenced by Ikev2DecodeSa(), and Ikev2EncodeSa().

UINT8 IKEV2_TRANSFORM::Reserved

UINT16 IKEV2_TRANSFORM::TransformId

Referenced by Ikev2DecodeSa(), and Ikev2EncodeSa().

UINT8 IKEV2_TRANSFORM::TransformType

Referenced by Ikev2DecodeSa(), and Ikev2EncodeSa().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_TRANSFORM_DATA Struct Reference

Data Fields

UINT8 TransformIndex
UINT8 TransformType
UINT16 TransformId
IKE_SA_ATTRIBUTE Attribute

Field Documentation

IKE_SA_ATTRIBUTE IKEV2_TRANSFORM_DATA::Attribute

Referenced by Ikev2DecodeSa(), Ikev2EncodeSa(), Ikev2InitializeSaData(), and Ikev2ParseProposalData().

UINT16 IKEV2_TRANSFORM_DATA::TransformId

Referenced by Ikev2DecodeSa(), Ikev2EncodeSa(), Ikev2InitializeSaData(), and Ikev2ParseProposalData().

UINT8 IKEV2_TRANSFORM_DATA::TransformIndex

Referenced by Ikev2InitializeSaData().

UINT8 IKEV2_TRANSFORM_DATA::TransformType

Referenced by Ikev2DecodeSa(), Ikev2EncodeSa(), Ikev2InitializeSaData(), and Ikev2ParseProposalData().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IKEV2_TS Struct Reference

Data Fields

IKEV2_COMMON_PAYLOAD_HEADER Header
UINT8 TSNumbers
UINT8 Reserved1
UINT16 Reserved2

Field Documentation

IKEV2_COMMON_PAYLOAD_HEADER IKEV2_TS::Header

Referenced by Ikev2GenerateTsPayload().

UINT8 IKEV2_TS::Reserved1

UINT16 IKEV2_TS::Reserved2

UINT8 IKEV2_TS::TSNumbers

Referenced by Ikev2DecodePayload(), Ikev2EncodePayload(), and Ikev2GenerateTsPayload().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IKEV2_VENDOR Struct Reference

Data Fields

IKEV2_COMMON_PAYLOAD_HEADER Header

Field Documentation

IKEV2_COMMON_PAYLOAD_HEADER IKEV2_VENDOR::Header

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

INSERT_POLICY_ENTRY_CONTEXT Struct Reference

Data Fields

EFI_IPSEC_CONFIG_DATA_TYPE	DataType
EFI_IPSEC_CONFIG_SELECTOR *	Selector
VOID *	Data
POLICY_ENTRY_INDEXER	Indexer
EFI_STATUS	Status

Field Documentation

VOID* INSERT_POLICY_ENTRY_CONTEXT::Data

Referenced by AddOrInsertPolicyEntry().

EFI_IPSEC_CONFIG_DATA_TYPE INSERT_POLICY_ENTRY_CONTEXT::DataType

Referenced by AddOrInsertPolicyEntry().

POLICY_ENTRY_INDEXER INSERT_POLICY_ENTRY_CONTEXT::Indexer

Referenced by AddOrInsertPolicyEntry().

EFI_IPSEC_CONFIG_SELECTOR* INSERT_POLICY_ENTRY_CONTEXT::Selector

Referenced by AddOrInsertPolicyEntry().

EFI_STATUS INSERT_POLICY_ENTRY_CONTEXT::Status

Referenced by AddOrInsertPolicyEntry().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IP6_ADDRESS_INFO_ENTRY Struct Reference

Data Fields

LIST_ENTRY Link
EFI_IP6_ADDRESS_INFO AddrInfo

Field Documentation

EFI_IP6_ADDRESS_INFO IP6_ADDRESS_INFO_ENTRY::AddrInfo

Referenced by Ip6BuildNvAddressInfo(), and Ip6ParseAddressListFromString().

LIST_ENTRY IP6_ADDRESS_INFO_ENTRY::Link

Referenced by Ip6FreeAddressInfoList(), and Ip6ParseAddressListFromString().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IP6_ASSEMBLE_ENTRY Struct Reference

Data Fields

LIST_ENTRY	Link
LIST_ENTRY	Fragments
EFI_IPv6_ADDRESS	Dst
EFI_IPv6_ADDRESS	Src
UINT32	Id
UINT32	TotalLen
UINT32	CurLen
UINT32	Life
EFI_IP6_HEADER *	Head
IP6_CLIP_INFO *	Info
NET_BUF *	Packet

Field Documentation

UINT32 IP6_ASSEMBLE_ENTRY::CurLen

Referenced by Ip6CreateAssembleEntry(), and Ip6Reassemble().

EFI_IPv6_ADDRESS IP6_ASSEMBLE_ENTRY::Dst

Referenced by Ip6CreateAssembleEntry(), and Ip6Reassemble().

LIST_ENTRY IP6_ASSEMBLE_ENTRY::Fragments

Referenced by Ip6CreateAssembleEntry(), and Ip6Reassemble().

EFI_IP6_HEADER* IP6_ASSEMBLE_ENTRY::Head

Referenced by Ip6CreateAssembleEntry(), Ip6PacketTimerTicking(), and Ip6Reassemble().

UINT32 IP6_ASSEMBLE_ENTRY::Id

Referenced by Ip6CreateAssembleEntry(), and Ip6Reassemble().

IP6_CLIP_INFO* IP6_ASSEMBLE_ENTRY::Info

Referenced by Ip6CreateAssembleEntry(), and Ip6Reassemble().

UINT32 IP6_ASSEMBLE_ENTRY::Life

Referenced by Ip6CreateAssembleEntry(), and Ip6PacketTimerTicking().

LIST_ENTRY IP6_ASSEMBLE_ENTRY::Link

Referenced by Ip6Reassemble().

NET_BUF* IP6_ASSEMBLE_ENTRY::Packet

Referenced by Ip6CreateAssembleEntry(), Ip6PacketTimerTicking(), and Ip6Reassemble().

EFI_IPv6_ADDRESS IP6_ASSEMBLE_ENTRY::Src

Referenced by Ip6CreateAssembleEntry(), and Ip6Reassemble().

UINT32 IP6_ASSEMBLE_ENTRY::TotalLen

Referenced by Ip6CreateAssembleEntry(), and Ip6Reassemble().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IP6_ASSEMBLE_TABLE Struct Reference

Data Fields

LIST_ENTRY Bucket [IP6_ASSEMLE_HASH_SIZE]

Field Documentation

LIST_ENTRY IP6_ASSEMBLE_TABLE::Bucket[IP6_ASSEMLE_HASH_SIZE]

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IP6_CLIP_INFO Struct Reference

Data Fields

UINT32	LinkFlag
INT32	CastType
INT32	Start
INT32	End
INT32	Length
UINT32	Life
EFI_STATUS	Status
UINT32	Id
UINT16	HeadLen
UINT8	NextHeader
UINT8	LastFrag
UINT32	FormerNextHeader

Field Documentation

INT32 IP6_CLIP_INFO::CastType

Referenced by Ip6InstanceFrameAcceptable(), Ip6InterfaceEnquePacket(), and Ip6PreProcessPacket().

INT32 IP6_CLIP_INFO::End

Referenced by Ip6PreProcessPacket(), Ip6Reassemble(), and Ip6TrimPacket().

UINT32 IP6_CLIP_INFO::FormerNextHeader

Referenced by Ip6PreProcessPacket(), and Ip6Reassemble().

UINT16 IP6_CLIP_INFO::HeadLen

Referenced by Ip6PreProcessPacket(), and Ip6Reassemble().

UINT32 IP6_CLIP_INFO::Id

Referenced by Ip6PreProcessPacket(), and Ip6Reassemble().

UINT8 IP6_CLIP_INFO::LastFrag

Referenced by Ip6PreProcessPacket(), and Ip6Reassemble().

INT32 IP6_CLIP_INFO::Length

Referenced by Ip6PreProcessPacket(), Ip6Reassemble(), and Ip6TrimPacket().

UINT32 IP6_CLIP_INFO::Life

Referenced by Ip6InstanceEnquePacket(), and Ip6PacketTimerTicking().

UINT32 IP6_CLIP_INFO::LinkFlag

Referenced by Ip6PreProcessPacket().

UINT8 IP6_CLIP_INFO::NextHeader

Referenced by Ip6PreProcessPacket(), and Ip6Reassemble().

INT32 IP6_CLIP_INFO::Start

Referenced by Ip6PreProcessPacket(), Ip6Reassemble(), and Ip6TrimPacket().

EFI_STATUS IP6_CLIP_INFO::Status

Referenced by Ip6PreProcessPacket().

IP6_CONFIG_DATA Union Reference

Data Fields

VOID *	Ptr
EFI_IP6_CONFIG_INTERFACE_INFO *	IfInfo
EFI_IP6_CONFIG_INTERFACE_ID *	AltIfId
EFI_IP6_CONFIG_POLICY *	Policy
EFI_IP6_CONFIG_DUP_ADDR_DETECT_TRANSMITS *	DadXmits
EFI_IP6_CONFIG_MANUAL_ADDRESS *	ManualAddress
EFI_IPv6_ADDRESS *	Gateway
EFI_IPv6_ADDRESS *	DnsServers

Field Documentation

EFI_IP6_CONFIG_INTERFACE_ID* IP6_CONFIG_DATA::AltIfId

Referenced by Ip6ConfigSetAltIfId().

EFI_IP6_CONFIG_DUP_ADDR_DETECT_TRANSMITS* IP6_CONFIG_DATA::DadXmits

EFI_IPv6_ADDRESS* IP6_CONFIG_DATA::DnsServers

Referenced by Ip6ConfigSetDnsServer().

EFI_IPv6_ADDRESS* IP6_CONFIG_DATA::Gateway

Referenced by Ip6ConfigSetGateway().

EFI_IP6_CONFIG_INTERFACE_INFO* IP6_CONFIG_DATA::IfInfo

EFI_IP6_CONFIG_MANUAL_ADDRESS* IP6_CONFIG_DATA::ManualAddress

Referenced by Ip6ManualAddrDadCallback().

EFI_IP6_CONFIG_POLICY* IP6_CONFIG_DATA::Policy

VOID* IP6_CONFIG_DATA::Ptr

Referenced by EfiIp6ConfigGetData(), Ip6ConfigCleanInstance(), Ip6ConfigGetIfInfo(), Ip6ConfigInitInstance(), Ip6ConfigParseDhcpReply(), Ip6ConfigReadConfigData(), Ip6ConfigSetDnsServer(), Ip6ConfigSetGateway(), Ip6ConfigSetMaunualAddress(), Ip6ConfigSetPolicy(),

Ip6ConfigWriteConfigData(), Ip6CreateService(), and Ip6ManualAddrDadCallback().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IP6_CONFIG_DATA_ITEM Struct Reference

Data Fields

IP6_CONFIG_SET_DATA	SetData
IP6_CONFIG_GET_DATA	GetData
EFI_STATUS	Status
UINT8	Attribute
NET_MAP	EventMap
IP6_CONFIG_DATA	Data
UINTN	DataSize

Field Documentation

UINT8 IP6_CONFIG_DATA_ITEM::Attribute

Referenced by Ip6ConfigCleanInstance(), Ip6ConfigInitInstance(), Ip6ConfigReadConfigData(), and Ip6ConfigWriteConfigData().

IP6_CONFIG_DATA IP6_CONFIG_DATA_ITEM::Data

Referenced by EfiIp6ConfigGetData(), Ip6ConfigCleanInstance(), Ip6ConfigGetIfInfo(), Ip6ConfigInitInstance(), Ip6ConfigParseDhcpReply(), Ip6ConfigReadConfigData(), Ip6ConfigSetAltIfId(), Ip6ConfigSetDnsServer(), Ip6ConfigSetGateway(), Ip6ConfigSetMaunualAddress(), Ip6ConfigSetPolicy(), Ip6ConfigWriteConfigData(), Ip6CreateService(), and Ip6ManualAddrDadCallback().

UINTN IP6_CONFIG_DATA_ITEM::DataSize

Referenced by EfiIp6ConfigGetData(), Ip6ConfigCleanInstance(), Ip6ConfigInitInstance(), Ip6ConfigParseDhcpReply(), Ip6ConfigReadConfigData(), Ip6ConfigSetDnsServer(), Ip6ConfigSetGateway(), Ip6ConfigSetMaunualAddress(), Ip6ConfigSetPolicy(), Ip6ConfigWriteConfigData(), Ip6CreateService(), and Ip6ManualAddrDadCallback().

NET_MAP IP6_CONFIG_DATA_ITEM::EventMap

Referenced by EfiIp6ConfigRegisterDataNotify(), EfiIp6ConfigSetData(), EfiIp6ConfigUnregisterDataNotify(), Ip6ConfigParseDhcpReply(), Ip6ConfigSetPolicy(), and Ip6ManualAddrDadCallback().

IP6_CONFIG_GET_DATA IP6_CONFIG_DATA_ITEM::GetData

Referenced by EfiIp6ConfigGetData(), and Ip6ConfigInitInstance().

IP6_CONFIG_SET_DATA IP6_CONFIG_DATA_ITEM::SetData

Referenced by EfiIp6ConfigSetData(), Ip6ConfigInitInstance(), and Ip6CreateService().

EFI_STATUS IP6_CONFIG_DATA_ITEM::Status

Referenced by EfiIp6ConfigGetData(), EfiIp6ConfigSetData(), Ip6ConfigInitInstance(), Ip6ConfigParseDhcpReply(), Ip6ConfigReadConfigData(), Ip6ConfigSetAltIfId(), Ip6ConfigSetDnsServer(), Ip6ConfigSetGateway(), Ip6ConfigSetMaunualAddress(), Ip6ConfigSetPolicy(), Ip6ConfigWriteConfigData(), and Ip6ManualAddrDadCallback().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IP6_CONFIG_DATA_RECORD Struct Reference

Data Fields

UINT16	Offset
UINT32	DataSize
EFI_IP6_CONFIG_DATA_TYPE	DataType

Field Documentation

UINT32 IP6_CONFIG_DATA_RECORD::DataSize

Referenced by Ip6ConfigReadConfigData(), and Ip6ConfigWriteConfigData().

EFI_IP6_CONFIG_DATA_TYPE IP6_CONFIG_DATA_RECORD::DataType

Referenced by Ip6ConfigReadConfigData(), and Ip6ConfigWriteConfigData().

UINT16 IP6_CONFIG_DATA_RECORD::Offset

Referenced by Ip6ConfigReadConfigData(), and Ip6ConfigWriteConfigData().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IP6_CONFIG_IFR_NVDATA Struct Reference

Data Fields

UINT8 IfType
interface type
UINT8 Padding [3]
UINT32 Policy
manual or automatic
UINT32 DadTransmitCount
dad transmits count
CHAR16 InterfaceId [INTERFACE_ID_STR_STORAGE]
alternative interface id
CHAR16 ManualAddress [ADDRESS_STR_MAX_SIZE]
IP addresses.
CHAR16 GatewayAddress [ADDRESS_STR_MAX_SIZE]
Gateway address.
CHAR16 DnsAddress [ADDRESS_STR_MAX_SIZE]
DNS server address.

Detailed Description

IP6_CONFIG_IFR_NVDATA contains the IP6 configure parameters for that NIC.

Field Documentation

UINT32 IP6_CONFIG_IFR_NVDATA::DadTransmitCount
dad transmits count

CHAR16 IP6_CONFIG_IFR_NVDATA::DnsAddress[ADDRESS_STR_MAX_SIZE]
DNS server address.

CHAR16 IP6_CONFIG_IFR_NVDATA::GatewayAddress[ADDRESS_STR_MAX_SIZE]
Gateway address.

UINT8 IP6_CONFIG_IFR_NVDATA::IfType
interface type

CHAR16 IP6_CONFIG_IFR_NVDATA::InterfaceId[INTERFACE_ID_STR_STORAGE]
alternative interface id

CHAR16 IP6_CONFIG_IFR_NVDATA::ManualAddress[ADDRESS_STR_MAX_SIZE]

IP addresses.

UINT8 IP6_CONFIG_IFR_NVDATA::Padding[3]

UINT32 IP6_CONFIG_IFR_NVDATA::Policy
manual or automatic

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IP6_CONFIG_NVDATA Struct Reference

Data Fields

EFI_IP6_CONFIG_POLICY	Policy manual or automatic
EFI_IP6_CONFIG_DUP_ADDR_DETECT_TRANSMITS	DadTransmitCount dad transmits count
EFI_IP6_CONFIG_INTERFACE_ID	InterfaceId alternative interface id
LIST_ENTRY	ManualAddress IP addresses.
UINT32	ManualAddressCount IP addresses count.
LIST_ENTRY	GatewayAddress Gateway address.
UINT32	GatewayAddressCount Gateway address count.
LIST_ENTRY	DnsAddress DNS server address.
UINT32	DnsAddressCount DNS server address count.

Field Documentation

EFI_IP6_CONFIG_DUP_ADDR_DETECT_TRANSMITS IP6_CONFIG_NVDATA::DadTransmitCount
dad transmits count

Referenced by Ip6ConvertIfrNvDataToConfigNvDataGeneral().

LIST_ENTRY IP6_CONFIG_NVDATA::DnsAddress
DNS server address.

Referenced by Ip6BuildNvAddressInfo(), Ip6ConfigFormUnload(), Ip6ConvertIfrNvDataToConfigNvDataAdvanced(), and Ip6FormCallback().

UINT32 IP6_CONFIG_NVDATA::DnsAddressCount
DNS server address count.

Referenced by Ip6BuildNvAddressInfo(), Ip6ConfigFormUnload(), Ip6ConvertIfrNvDataToConfigNvDataAdvanced(), and Ip6FormCallback().

LIST_ENTRY IP6_CONFIG_NVDATA::GatewayAddress
Gateway address.

Referenced by Ip6BuildNvAddressInfo(), Ip6ConfigFormUnload(), Ip6ConvertIfrNvDataToConfigNvDataAdvanced(), and Ip6FormCallback().

UINT32 IP6_CONFIG_NVDATA::GatewayAddressCount
Gateway address count.

Referenced by Ip6BuildNvAddressInfo(), Ip6ConfigFormUnload(), Ip6ConvertIfrNvDataToConfigNvDataAdvanced(), and Ip6FormCallback().

EFI_IP6_CONFIG_INTERFACE_ID IP6_CONFIG_NVDATA::InterfaceId
alternative interface id

Referenced by Ip6ConvertConfigNvDataToIfrNvData(), Ip6ConvertIfrNvDataToConfigNvDataGeneral(), and Ip6FormCallback().

LIST_ENTRY IP6_CONFIG_NVDATA::ManualAddress
IP addresses.

Referenced by Ip6BuildNvAddressInfo(), Ip6ConfigFormUnload(), Ip6ConvertIfrNvDataToConfigNvDataAdvanced(), and Ip6FormCallback().

UINT32 IP6_CONFIG_NVDATA::ManualAddressCount
IP addresses count.

Referenced by Ip6BuildNvAddressInfo(), Ip6ConfigFormUnload(), Ip6ConvertIfrNvDataToConfigNvDataAdvanced(), and Ip6FormCallback().

EFI_IP6_CONFIG_POLICY IP6_CONFIG_NVDATA::Policy
manual or automatic

Referenced by Ip6ConvertIfrNvDataToConfigNvDataAdvanced(), and Ip6ConvertIfrNvDataToConfigNvDataGeneral().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IP6_CONFIG_VARIABLE Struct Reference

Data Fields

UINT32 IaId
UINT16 Checksum
UINT16 DataRecordCount
IP6_CONFIG_DATA_RECORD DataRecord [1]

Field Documentation

UINT16 IP6_CONFIG_VARIABLE::Checksum

Referenced by Ip6ConfigWriteConfigData().

IP6_CONFIG_DATA_RECORD IP6_CONFIG_VARIABLE::DataRecord[1]

Referenced by Ip6ConfigWriteConfigData().

UINT16 IP6_CONFIG_VARIABLE::DataRecordCount

Referenced by Ip6ConfigWriteConfigData().

UINT32 IP6_CONFIG_VARIABLE::IaId

Referenced by Ip6ConfigWriteConfigData().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IP6_DESTROY_CHILD_BY_ADDR_CALLBACK_CONTEXT Struct Reference

Data Fields

EFI_SERVICE_BINDING_PROTOCOL * ServiceBinding
EFI_IPv6_ADDRESS * Address

Field Documentation

EFI_IPv6_ADDRESS* IP6_DESTROY_CHILD_BY_ADDR_CALLBACK_CONTEXT::Address

Referenced by Ip6DestroyInstanceByAddress().

EFI_SERVICE_BINDING_PROTOCOL*
IP6_DESTROY_CHILD_BY_ADDR_CALLBACK_CONTEXT::ServiceBinding

Referenced by Ip6DestroyInstanceByAddress().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT Struct Reference

Data Fields

EFI_SERVICE_BINDING_PROTOCOL * ServiceBinding
UINTN NumberOfChildren
EFI_HANDLE * ChildHandleBuffer

Field Documentation

EFI_HANDLE* IP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer

Referenced by Ip6DriverBindingStop().

UINTN IP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren

Referenced by Ip6DriverBindingStop().

EFI_SERVICE_BINDING_PROTOCOL*
IP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding

Referenced by Ip6DriverBindingStop().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IP6_IPSEC_WRAP Struct Reference

Data Fields

EFI_EVENT IpSecRecycleSignal
NET_BUF * Packet

Field Documentation

EFI_EVENT IP6_IPSEC_WRAP::IpSecRecycleSignal

Referenced by Ip6IpSecFree(), and Ip6IpSecProcessPacket().

NET_BUF* IP6_IPSEC_WRAP::Packet

Referenced by Ip6IpSecFree(), and Ip6IpSecProcessPacket().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IP6_LINK_RX_TOKEN Struct Reference

Data Fields

UINT32	Signature
IP6_FRAME_CALLBACK	CallBack
VOID *	Context
EFI_MANAGED_NETWORK_COMPLETION_TOKEN	MnpToken

Field Documentation

IP6_FRAME_CALLBACK IP6_LINK_RX_TOKEN::CallBack

Referenced by Ip6CreateService(), Ip6OnFrameReceivedDpc(), and Ip6ReceiveFrame().

VOID* IP6_LINK_RX_TOKEN::Context

Referenced by Ip6CreateService(), Ip6OnFrameReceivedDpc(), and Ip6ReceiveFrame().

EFI_MANAGED_NETWORK_COMPLETION_TOKEN IP6_LINK_RX_TOKEN::MnpToken

Referenced by Ip6CreateService(), Ip6OnFrameReceivedDpc(), and Ip6ReceiveFrame().

UINT32 IP6_LINK_RX_TOKEN::Signature

Referenced by Ip6CreateService().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IP6_LINK_TX_TOKEN Struct Reference

Data Fields

UINT32	Signature
LIST_ENTRY	Link
IP6_PROTOCOL *	IpInstance
IP6_FRAME_CALLBACK	CallBack
NET_BUF *	Packet
VOID *	Context
EFI_MAC_ADDRESS	DstMac
EFI_MAC_ADDRESS	SrcMac
EFI_MANAGED_NETWORK_COMPLETION_TOKEN	MnpToken
EFI_MANAGED_NETWORK_TRANSMIT_DATA	MnpTxData

Field Documentation

IP6_FRAME_CALLBACK IP6_LINK_TX_TOKEN::CallBack

Referenced by Ip6CreateLinkTxToken(), Ip6FreeNeighborEntry(), Ip6OnArpResolved(), and Ip6OnFrameSentDpc().

VOID* IP6_LINK_TX_TOKEN::Context

Referenced by Ip6CreateLinkTxToken(), Ip6FreeNeighborEntry(), Ip6OnArpResolved(), and Ip6OnFrameSentDpc().

EFI_MAC_ADDRESS IP6_LINK_TX_TOKEN::DstMac

Referenced by Ip6CreateLinkTxToken(), Ip6OnArpResolved(), and Ip6SendFrame().

IP6_PROTOCOL* IP6_LINK_TX_TOKEN::IpInstance

Referenced by Ip6CreateLinkTxToken().

LIST_ENTRY IP6_LINK_TX_TOKEN::Link

Referenced by Ip6CreateLinkTxToken(), Ip6OnArpResolved(), Ip6OnFrameSentDpc(), and Ip6SendFrame().

EFI_MANAGED_NETWORK_COMPLETION_TOKEN IP6_LINK_TX_TOKEN::MnpToken

Referenced by Ip6CancelFrames(), Ip6CreateLinkTxToken(), Ip6OnArpResolved(), Ip6OnFrameSentDpc(), and Ip6SendFrame().

EFI_MANAGED_NETWORK_TRANSMIT_DATA IP6_LINK_TX_TOKEN::MnpTxData

Referenced by Ip6CreateLinkTxToken().

NET_BUF* IP6_LINK_TX_TOKEN::Packet

Referenced by Ip6CreateLinkTxToken(), Ip6FreeNeighborEntry(), Ip6OnArpResolved(), and Ip6OnFrameSentDpc().

UINT32 IP6_LINK_TX_TOKEN::Signature

Referenced by Ip6CreateLinkTxToken().

EFI_MAC_ADDRESS IP6_LINK_TX_TOKEN::SrcMac

Referenced by Ip6CreateLinkTxToken().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IP6_MLD_GROUP Struct Reference

Data Fields

LIST_ENTRY	Link
INTN	RefCnt
EFI_IPv6_ADDRESS	Address
UINT32	DelayTimer
BOOLEAN	SendByUs
EFI_MAC_ADDRESS	Mac

Field Documentation

EFI_IPv6_ADDRESS IP6_MLD_GROUP::Address

Referenced by Ip6CreateMldEntry(), Ip6FindMldEntry(), Ip6MldTimerTicking(), and Ip6ProcessMldQuery().

UINT32 IP6_MLD_GROUP::DelayTimer

Referenced by Ip6CreateMldEntry(), Ip6MldTimerTicking(), and Ip6ProcessMldReport().

LIST_ENTRY IP6_MLD_GROUP::Link

Referenced by Ip6CreateMldEntry(), Ip6InitMld(), Ip6JoinGroup(), and Ip6LeaveGroup().

EFI_MAC_ADDRESS IP6_MLD_GROUP::Mac

Referenced by Ip6FindMac(), Ip6InitMld(), Ip6JoinGroup(), and Ip6LeaveGroup().

INTN IP6_MLD_GROUP::RefCnt

Referenced by Ip6CreateMldEntry(), Ip6JoinGroup(), and Ip6LeaveGroup().

BOOLEAN IP6_MLD_GROUP::SendByUs

Referenced by Ip6CreateMldEntry(), Ip6JoinGroup(), Ip6LeaveGroup(), and Ip6ProcessMldReport().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Data Fields

IP6_MLD_HEAD Struct Reference

Data Fields

IP6_ICMP_HEAD Head
UINT16 MaxRespDelay
UINT16 Reserved
EFI_IPv6_ADDRESS Group

Field Documentation

EFI_IPv6_ADDRESS IP6_MLD_HEAD::Group

Referenced by Ip6SendMldDone(), and Ip6SendMldReport().

IP6_ICMP_HEAD IP6_MLD_HEAD::Head

Referenced by Ip6SendMldDone(), and Ip6SendMldReport().

UINT16 IP6_MLD_HEAD::MaxRespDelay

UINT16 IP6_MLD_HEAD::Reserved

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IP6_MLD_SERVICE_DATA Struct Reference

Data Fields

INTN Mldv1QuerySeen
LIST_ENTRY Groups

Field Documentation

LIST_ENTRY IP6_MLD_SERVICE_DATA::Groups

Referenced by Ip6CreateService().

INTN IP6_MLD_SERVICE_DATA::Mldv1QuerySeen

Referenced by Ip6CreateService().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IP6_ROUTE_CACHE Struct Reference

Data Fields

LIST_ENTRY CacheBucket [IP6_ROUTE_CACHE_HASH_SIZE]
UINT8 CacheNum [IP6_ROUTE_CACHE_HASH_SIZE]

Field Documentation

LIST_ENTRY IP6_ROUTE_CACHE::CacheBucket[IP6_ROUTE_CACHE_HASH_SIZE]

Referenced by Ip6CreateRouteTable(), and Ip6Route().

UINT8 IP6_ROUTE_CACHE::CacheNum[IP6_ROUTE_CACHE_HASH_SIZE]

Referenced by Ip6CreateRouteTable(), and Ip6Route().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IP6_ROUTE_CACHE_ENTRY Struct Reference

Data Fields

LIST_ENTRY	Link
INTN	RefCnt
UINTN	Tag
EFI_IPv6_ADDRESS	Destination
EFI_IPv6_ADDRESS	Source
EFI_IPv6_ADDRESS	NextHop

Field Documentation

EFI_IPv6_ADDRESS IP6_ROUTE_CACHE_ENTRY::Destination

Referenced by Ip6CreateRouteCacheEntry(), and Ip6FindRouteCache().

LIST_ENTRY IP6_ROUTE_CACHE_ENTRY::Link

Referenced by Ip6ProcessRedirect(), and Ip6Route().

EFI_IPv6_ADDRESS IP6_ROUTE_CACHE_ENTRY::NextHop

Referenced by Ip6CreateRouteCacheEntry(), Ip6Output(), and Ip6ProcessRedirect().

INTN IP6_ROUTE_CACHE_ENTRY::RefCnt

Referenced by Ip6CreateRouteCacheEntry().

EFI_IPv6_ADDRESS IP6_ROUTE_CACHE_ENTRY::Source

Referenced by Ip6CreateRouteCacheEntry(), and Ip6FindRouteCache().

UINTN IP6_ROUTE_CACHE_ENTRY::Tag

Referenced by Ip6CreateRouteCacheEntry(), Ip6ProcessRedirect(), and Ip6PurgeRouteCache().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Data Fields

IP6_ROUTE_ENTRY Struct Reference

Data Fields

LIST_ENTRY	Link
INTN	RefCnt
UINT32	Flag
UINT8	PrefixLength
EFI_IPv6_ADDRESS	Destination
EFI_IPv6_ADDRESS	NextHop

Field Documentation

EFI_IPv6_ADDRESS IP6_ROUTE_ENTRY::Destination

Referenced by Ip6AddRoute(), Ip6BuildEfiRouteTable(), Ip6CreateRouteEntry(), Ip6DelRoute(), and Ip6FindRouteEntry().

UINT32 IP6_ROUTE_ENTRY::Flag

Referenced by Ip6AddRoute(), Ip6CreatePrefixListEntry(), Ip6CreateRouteEntry(), Ip6Output(), Ip6ProcessPacketTooBig(), Ip6ProcessRedirect(), and Ip6Route().

LIST_ENTRY IP6_ROUTE_ENTRY::Link

Referenced by Ip6AddRoute(), Ip6CreateDefaultRouter(), Ip6CreatePrefixListEntry(), and Ip6ProcessPacketTooBig().

EFI_IPv6_ADDRESS IP6_ROUTE_ENTRY::NextHop

Referenced by Ip6AddRoute(), Ip6BuildEfiRouteTable(), Ip6CreateRouteEntry(), Ip6DelRoute(), Ip6FindRouteEntry(), and Ip6Route().

UINT8 IP6_ROUTE_ENTRY::PrefixLength

Referenced by Ip6BuildEfiRouteTable(), Ip6CreateRouteEntry(), and Ip6FindRouteEntry().

INTN IP6_ROUTE_ENTRY::RefCnt

Referenced by Ip6CreateRouteEntry().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page

- Data Structures
- Files

- Data Structures
- Data Fields

IP6_RXDATA_WRAP Struct Reference

Data Fields

LIST_ENTRY Link
IP6_PROTOCOL * IpInstance
NET_BUF * Packet
EFI_IP6_RECEIVE_DATA RxData

Field Documentation

IP6_PROTOCOL* IP6_RXDATA_WRAP::IpInstance

Referenced by Ip6OnRecyclePacket(), and Ip6WrapRxData().

LIST_ENTRY IP6_RXDATA_WRAP::Link

Referenced by Ip6InstanceDeliverPacket(), Ip6OnRecyclePacket(), and Ip6WrapRxData().

NET_BUF* IP6_RXDATA_WRAP::Packet

Referenced by Ip6OnRecyclePacket(), and Ip6WrapRxData().

EFI_IP6_RECEIVE_DATA IP6_RXDATA_WRAP::RxData

Referenced by Ip6InstanceDeliverPacket(), Ip6OnRecyclePacket(), and Ip6WrapRxData().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IP6_TXTOKEN_WRAP Struct Reference

Data Fields

IP6_PROTOCOL *	IpInstance
EFI_IP6_COMPLETION_TOKEN *	Token
EFI_EVENT	IpSecRecycleSignal
NET_BUF *	Packet
BOOLEAN	Sent
INTN	Life

Field Documentation

IP6_PROTOCOL* IP6_TXTOKEN_WRAP::IpInstance

Referenced by EfiIp6Transmit(), Ip6CancelTxTokens(), Ip6FreeTxToken(), and Ip6SentPacketTicking().

EFI_EVENT IP6_TXTOKEN_WRAP::IpSecRecycleSignal

Referenced by Ip6FreeTxToken(), and Ip6IpSecProcessPacket().

INTN IP6_TXTOKEN_WRAP::Life

Referenced by EfiIp6Transmit(), and Ip6SentPacketTicking().

NET_BUF* IP6_TXTOKEN_WRAP::Packet

Referenced by EfiIp6Transmit(), Ip6CancelTxTokens(), Ip6IpSecProcessPacket(), Ip6OnPacketSent(), and Ip6SentPacketTicking().

BOOLEAN IP6_TXTOKEN_WRAP::Sent

Referenced by EfiIp6Transmit(), and Ip6FreeTxToken().

EFI_IP6_COMPLETION_TOKEN* IP6_TXTOKEN_WRAP::Token

Referenced by EfiIp6Transmit(), Ip6FreeTxToken(), Ip6OnPacketSent(), and Ip6Output().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Data Structures
- Data Fields

IP_SEC_VARIABLE_INFO Struct Reference

Data Fields

UINT32 VariableCount
UINT32 VariableSize
UINT32 SingleVariableSize

Field Documentation

UINT32 IP_SEC_VARIABLE_INFO::SingleVariableSize

Referenced by IpSecGetVariable(), and IpSecSetVariable().

UINT32 IP_SEC_VARIABLE_INFO::VariableCount

Referenced by IpSecGetVariable(), and IpSecSetVariable().

UINT32 IP_SEC_VARIABLE_INFO::VariableSize

Referenced by IpSecGetVariable(), and IpSecSetVariable().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

IPSEC_COMMON_POLICY_ENTRY Struct Reference

Data Fields

EFI_IPSEC_CONFIG_SELECTOR * Selector
VOID * Data
LIST_ENTRY List

Field Documentation

VOID* IPSEC_COMMON_POLICY_ENTRY::Data

LIST_ENTRY IPSEC_COMMON_POLICY_ENTRY::List

EFI_IPSEC_CONFIG_SELECTOR* IPSEC_COMMON_POLICY_ENTRY::Selector

Referenced by EfiIpSecConfigGetNextSelector().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IPSEC_VAR_ITEM_HEADER Struct Reference

Data Fields

UINT8 Type
UINT16 Size

Field Documentation

UINT16 IPSEC_VAR_ITEM_HEADER::Size

Referenced by IpSecConfigRestore(), and IpSecCopyPolicyEntry().

UINT8 IPSEC_VAR_ITEM_HEADER::Type

Referenced by IpSecConfigRestore(), and IpSecCopyPolicyEntry().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

IPSEC_VARIABLE_BUFFER Struct Reference

Data Fields

UINT8 * Ptr
UINTN Size
UINTN Capacity

Field Documentation

UINTN IPSEC_VARIABLE_BUFFER::Capacity

Referenced by IpSecConfigSave().

UINT8* IPSEC_VARIABLE_BUFFER::Ptr

Referenced by IpSecConfigSave().

UINTN IPSEC_VARIABLE_BUFFER::Size

Referenced by IpSecConfigSave().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

ISCSI_NIC_INFO Struct Reference

Data Fields

LIST_ENTRY	Link
UINT32	HwAddressSize
EFI_MAC_ADDRESS	PermanentAddress
UINT8	NicIndex
UINT16	VlanId
UINTN	BusNumber
UINTN	DeviceNumber
UINTN	FunctionNumber

Field Documentation

UINTN ISCSI_NIC_INFO::BusNumber

Referenced by IScsiAddNic(), IScsiConfigAddAttempt(), IScsiConfigProcessDefault(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiFillNICAndTargetSections(), and IScsiGetConfigData().

UINTN ISCSI_NIC_INFO::DeviceNumber

Referenced by IScsiAddNic(), IScsiConfigAddAttempt(), IScsiConfigProcessDefault(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiFillNICAndTargetSections(), and IScsiGetConfigData().

UINTN ISCSI_NIC_INFO::FunctionNumber

Referenced by IScsiAddNic(), IScsiConfigAddAttempt(), IScsiConfigProcessDefault(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiFillNICAndTargetSections(), and IScsiGetConfigData().

UINT32 ISCSI_NIC_INFO::HwAddressSize

Referenced by IScsiAddNic(), IScsiConfigAddAttempt(), IScsiConfigProcessDefault(), IScsiGetConfigData(), and IScsiRemoveNic().

LIST_ENTRY ISCSI_NIC_INFO::Link

Referenced by IScsiAddNic(), and IScsiRemoveNic().

UINT8 ISCSI_NIC_INFO::NicIndex

Referenced by IScsiAddNic(), IScsiConfigAddAttempt(), IScsiGetConfigData(), IScsiGetNicInfoByIndex(), and IScsiRemoveNic().

EFI_MAC_ADDRESS ISCSI_NIC_INFO::PermanentAddress

Referenced by IScsiAddNic(), IScsiConfigAddAttempt(), IScsiConfigProcessDefault(), IScsiFillNICAndTargetSections(), IScsiGetConfigData(), and IScsiRemoveNic().

UINT16 ISCSI_NIC_INFO::VlanId

Referenced by IScsiAddNic(), IScsiConfigAddAttempt(), IScsiConfigProcessDefault(), IScsiFillNICAndTargetSections(), IScsiGetConfigData(), and IScsiRemoveNic().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

ISCSI_PRIVATE_DATA Struct Reference

Data Fields

```
      CHAR16 PortString [ISCSI_NAME_IFR_MAX_SIZE]
LIST_ENTRY NicInfoList
       UINT8 NicCount
       UINT8 CurrentNic
       UINT8 MaxNic
     BOOLEAN Ipv6Flag
     BOOLEAN OneSessionEstablished
     BOOLEAN EnableMpio
       UINT8 MpioCount
       UINT8 Krb5MpioCount
       UINT8 SinglePathCount
       UINT8 ValidSinglePathCount
       UINT8 BootSelectedIndex
       UINT8 AttemptCount
LIST_ENTRY AttemptConfigs
       CHAR8 InitiatorName [ISCSI_NAME_MAX_SIZE]
       UINTN InitiatorNameLength
       VOID * NewAttempt
```

Field Documentation

LIST_ENTRY ISCSI_PRIVATE_DATA::AttemptConfigs

Referenced by IScsiConfigDeleteAttempts(), IScsiConfigDisplayDeleteAttempts(), IScsiConfigDisplayOrderAttempts(), IScsiConfigFormUnload(), IScsiConfigGetAttemptByConfigIndex(), IScsiConfigGetAttemptByNic(), IScsiConfigOrderAttempts(), IScsiConfigUpdateAttempt(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiDriverEntryPoint(), IScsiFillNICAndTargetSections(), IScsiGetConfigData(), IScsiRemoveNic(), and IScsiStart().

UINT8 ISCSI_PRIVATE_DATA::AttemptCount

Referenced by IScsiConfigDeleteAttempts(), IScsiConfigDisplayOrderAttempts(), IScsiConfigFormUnload(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiGetConfigData(), and IScsiRemoveNic().

UINT8 ISCSI_PRIVATE_DATA::BootSelectedIndex

Referenced by IScsiFillNICAndTargetSections(), and IScsiStart().

UINT8 ISCSI_PRIVATE_DATA::CurrentNic

Referenced by IScsiAddNic(), IScsiGetConfigData(), IScsiRemoveNic(), and IScsiStart().

BOOLEAN ISCSI_PRIVATE_DATA::EnableMpio

Referenced by IScsiConfigDeleteAttempts(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiFillNICAndTargetSections(), IScsiGetConfigData(), IScsiInitControlSection(), IScsiPublishIbft(), IScsiRemoveNic(), and IScsiStart().

CHAR8 ISCSI_PRIVATE_DATA::InitiatorName[ISCSI_NAME_MAX_SIZE]

Referenced by IScsiCHAPToSendReq(), IScsiFillInitiatorSection(), IScsiFormCallback(), and IScsiGetConfigData().

UINTN ISCSI_PRIVATE_DATA::InitiatorNameLength

Referenced by IScsiFillInitiatorSection(), IScsiFormCallback(), and IScsiGetConfigData().

BOOLEAN ISCSI_PRIVATE_DATA::Ipv6Flag

Referenced by IScsiGetConfigData(), and IScsiStart().

UINT8 ISCSI_PRIVATE_DATA::Krb5MpioCount

Referenced by IScsiGetConfigData(), IScsiInitControlSection(), IScsiPublishIbft(), and IScsiRemoveNic().

UINT8 ISCSI_PRIVATE_DATA::MaxNic

Referenced by IScsiAddNic(), and IScsiConfigProcessDefault().

UINT8 ISCSI_PRIVATE_DATA::MpioCount

Referenced by IScsiConfigDeleteAttempts(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiGetConfigData(), IScsiInitControlSection(), IScsiPublishIbft(), and IScsiRemoveNic().

VOID* ISCSI_PRIVATE_DATA::NewAttempt

Referenced by IScsiConfigFormUnload(), IScsiConfigProcessDefault(), IScsiConvertIfrNvDataToAttemptConfigData(), and IScsiRemoveNic().

UINT8 ISCSI_PRIVATE_DATA::NicCount

Referenced by IScsiAddNic(), IScsiConfigFormUnload(), and IScsiRemoveNic().

LIST_ENTRY ISCSI_PRIVATE_DATA::NicInfoList

Referenced by IScsiAddNic(), IScsiConfigAddAttempt(), IScsiConfigFormUnload(), IScsiDriverEntryPoint(), IScsiGetNicInfoByIndex(), and IScsiRemoveNic().

BOOLEAN ISCSI_PRIVATE_DATA::OneSessionEstablished

Referenced by IScsiCleanDriverData(), and IScsiStart().

CHAR16 ISCSI_PRIVATE_DATA::PortString[ISCSI_NAME_IFR_MAX_SIZE]

Referenced by IScsiConfigAddAttempt(), IScsiConfigDeleteAttempts(), IScsiConfigProcessDefault(), IScsiConfigUpdateAttempt(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiGetConfigData(), and IScsiStart().

UINT8 ISCSI_PRIVATE_DATA::SinglePathCount

Referenced by IScsiConfigDeleteAttempts(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiGetConfigData(), and IScsiRemoveNic().

UINT8 ISCSI_PRIVATE_DATA::ValidSinglePathCount

Referenced by IScsiGetConfigData(), IScsiInitControlSection(), IScsiPublishIbft(), IScsiRemoveNic(), and IScsiStart().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

MODP_GROUP Struct Reference

Data Fields

UINT8 GroupId
UINTN Size
UINT8 * Modulus
UINTN GroupGenerator

Field Documentation

UINTN MODP_GROUP::GroupGenerator

Referenced by Ikev2GenerateSaDhPublicKey().

UINT8 MODP_GROUP::GroupId

UINT8* MODP_GROUP::Modulus

Referenced by Ikev2GenerateSaDhPublicKey().

UINTN MODP_GROUP::Size

Referenced by Ikev2GenerateSaDhPublicKey().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

MTFTP6_BLOCK_RANGE Struct Reference

Data Fields

LIST_ENTRY Link
INTN Start
INTN End
INTN Round
INTN Bound

Field Documentation

INTN MTFTP6_BLOCK_RANGE::Bound

Referenced by Mtftp6AllocateRange(), and Mtftp6RemoveBlockNum().

INTN MTFTP6_BLOCK_RANGE::End

Referenced by Mtftp6AllocateRange(), Mtftp6RemoveBlockNum(), and Mtftp6SetLastBlockNum().

LIST_ENTRY MTFTP6_BLOCK_RANGE::Link

Referenced by Mtftp6AllocateRange(), Mtftp6InitBlockRange(), Mtftp6RemoveBlockNum(), and Mtftp6SetLastBlockNum().

INTN MTFTP6_BLOCK_RANGE::Round

Referenced by Mtftp6RemoveBlockNum().

INTN MTFTP6_BLOCK_RANGE::Start

Referenced by Mtftp6AllocateRange(), Mtftp6GetNextBlockNum(), Mtftp6RemoveBlockNum(), and Mtftp6SetLastBlockNum().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

MTFTP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEX
Struct Reference

Data Fields

EFI_SERVICE_BINDING_PROTOCOL * ServiceBinding
UINTN NumberOfChildren
EFI_HANDLE * ChildHandleBuffer

Field Documentation

EFI_HANDLE* MTFTP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer

Referenced by Mtftp6DriverBindingStop().

UINTN MTFTP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren

Referenced by Mtftp6DriverBindingStop().

EFI_SERVICE_BINDING_PROTOCOL*
MTFTP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding

Referenced by Mtftp6DriverBindingStop().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

MTFTP6_EXT_OPTION_INFO Struct Reference

Data Fields

UINT16	BlkSize
UINT8	Timeout
UINT32	Tsize
EFI_IPv6_ADDRESS	McastIp
UINT16	McastPort
BOOLEAN	IsMaster
UINT32	BitMap

Field Documentation

UINT32 MTFTP6_EXT_OPTION_INFO::BitMap

Referenced by Mtftp6RrqHandleOack().

UINT16 MTFTP6_EXT_OPTION_INFO::BlkSize

Referenced by Mtftp6RrqHandleOack(), and Mtftp6WrqHandleOack().

BOOLEAN MTFTP6_EXT_OPTION_INFO::IsMaster

Referenced by Mtftp6RrqHandleOack().

EFI_IPv6_ADDRESS MTFTP6_EXT_OPTION_INFO::McastIp

Referenced by Mtftp6RrqHandleOack().

UINT16 MTFTP6_EXT_OPTION_INFO::McastPort

Referenced by Mtftp6RrqHandleOack().

UINT8 MTFTP6_EXT_OPTION_INFO::Timeout

Referenced by Mtftp6RrqHandleOack(), and Mtftp6WrqHandleOack().

UINT32 MTFTP6_EXT_OPTION_INFO::Tsize

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page

- Data Structures
- Files

- Data Structures
- Data Fields

PAD_ENTRY_INDEXER Struct Reference

Data Fields

EFI_IPSEC_PAD_ID PadId
UINTN Index

Field Documentation

UINTN PAD_ENTRY_INDEXER::Index

EFI_IPSEC_PAD_ID PAD_ENTRY_INDEXER::PadId

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

POLICY_ENTRY_INDEXER Union Reference

Data Fields

SPD_ENTRY_INDEXER Spd
SAD_ENTRY_INDEXER Sad
PAD_ENTRY_INDEXER Pad

Field Documentation

PAD_ENTRY_INDEXER POLICY_ENTRY_INDEXER::Pad

SAD_ENTRY_INDEXER POLICY_ENTRY_INDEXER::Sad

SPD_ENTRY_INDEXER POLICY_ENTRY_INDEXER::Spd

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

PRF_DATA_FRAGMENT Struct Reference

Data Fields

UINTN DataSize
UINT8 * Data

Field Documentation

UINT8* PRF_DATA_FRAGMENT::Data

Referenced by Ikev2CertGenerateAuthPayload(), Ikev2GenerateChildSaKeys(), Ikev2GenerateSaKeys(), Ikev2PskGenerateAuthPayload(), and Ikev2SaGenerateKey().

UINTN PRF_DATA_FRAGMENT::DataSize

Referenced by Ikev2CertGenerateAuthPayload(), Ikev2GenerateChildSaKeys(), Ikev2GenerateSaKeys(), Ikev2PskGenerateAuthPayload(), and Ikev2SaGenerateKey().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

PXEBC_BOOT_MENU_ENTRY Struct Reference

Data Fields

UINT16 Type
UINT8 DescLen
UINT8 DescStr [1]

Field Documentation

UINT8 PXEBC_BOOT_MENU_ENTRY::DescLen

Referenced by PxeBcSelectBootMenu().

UINT8 PXEBC_BOOT_MENU_ENTRY::DescStr[1]

UINT16 PXEBC_BOOT_MENU_ENTRY::Type

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

PXEBC_BOOT_SVR_ENTRY Struct Reference

Data Fields

UINT16 Type
UINT8 IpCnt
EFI_IPv4_ADDRESS IpAddr [1]

Field Documentation

EFI_IPv4_ADDRESS PXEBC_BOOT_SVR_ENTRY::IpAddr[1]

Referenced by EfiPxeBcDiscover(), PxeBcDhcp4BootInfo(), and PxeBcExtractDiscoverInfo().

UINT8 PXEBC_BOOT_SVR_ENTRY::IpCnt

Referenced by PxeBcDhcp4BootInfo(), and PxeBcExtractDiscoverInfo().

UINT16 PXEBC_BOOT_SVR_ENTRY::Type

Referenced by PxeBcExtractDiscoverInfo().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

PXEBC_CLASS_ID Struct Reference

Data Fields

UINT8 ClassIdentifier [10]
UINT8 ArchitecturePrefix [5]
UINT8 ArchitectureType [5]
UINT8 Lit3 [1]
UINT8 InterfaceName [4]
UINT8 Lit4 [1]
UINT8 UndiMajor [3]
UINT8 UndiMinor [3]

Field Documentation

UINT8 PXEBC_CLASS_ID::ArchitecturePrefix[5]

UINT8 PXEBC_CLASS_ID::ArchitectureType[5]

Referenced by PxeBcBuildDhcp6Options().

UINT8 PXEBC_CLASS_ID::ClassIdentifier[10]

UINT8 PXEBC_CLASS_ID::InterfaceName[4]

Referenced by PxeBcBuildDhcp6Options().

UINT8 PXEBC_CLASS_ID::Lit3[1]

UINT8 PXEBC_CLASS_ID::Lit4[1]

UINT8 PXEBC_CLASS_ID::UndiMajor[3]

Referenced by PxeBcBuildDhcp6Options().

UINT8 PXEBC_CLASS_ID::UndiMinor[3]

Referenced by PxeBcBuildDhcp6Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page

- Data Structures
- Files

- Data Structures
- Data Fields

PXEBC_DHCP4_OPTION_ARCH Struct Reference

Data Fields

UINT16 Type

Field Documentation

UINT16 PXEBC_DHCP4_OPTION_ARCH::Type

Referenced by PxeBcBuildDhcp4Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

PXEBC_DHCP4_OPTION_CLID Struct Reference

Data Fields

UINT8 ClassIdentifier [10]
UINT8 ArchitecturePrefix [5]
UINT8 ArchitectureType [5]
UINT8 Lit3 [1]
UINT8 InterfaceName [4]
UINT8 Lit4 [1]
UINT8 UndiMajor [3]
UINT8 UndiMinor [3]

Field Documentation

UINT8 PXEBC_DHCP4_OPTION_CLID::ArchitecturePrefix[5]

UINT8 PXEBC_DHCP4_OPTION_CLID::ArchitectureType[5]

Referenced by PxeBcBuildDhcp4Options().

UINT8 PXEBC_DHCP4_OPTION_CLID::ClassIdentifier[10]

UINT8 PXEBC_DHCP4_OPTION_CLID::InterfaceName[4]

Referenced by PxeBcBuildDhcp4Options().

UINT8 PXEBC_DHCP4_OPTION_CLID::Lit3[1]

UINT8 PXEBC_DHCP4_OPTION_CLID::Lit4[1]

UINT8 PXEBC_DHCP4_OPTION_CLID::UndiMajor[3]

Referenced by PxeBcBuildDhcp4Options().

UINT8 PXEBC_DHCP4_OPTION_CLID::UndiMinor[3]

Referenced by PxeBcBuildDhcp4Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page

- Data Structures
- Files

- Data Structures
- Data Fields

PXEBC_DHCP4_OPTION_ENTRY Union Reference

Data Fields

PXEBC_DHCP4_OPTION_PARA * Para
PXEBC_DHCP4_OPTION_UNDI * Undi
PXEBC_DHCP4_OPTION_ARCH * Arch
PXEBC_DHCP4_OPTION_CLID * Clid
PXEBC_DHCP4_OPTION_UUID * Uuid
PXEBC_DHCP4_OPTION_MESG * Mesg
PXEBC_DHCP4_OPTION_MAX_MESG_SIZE * MaxMesgSize

Field Documentation

PXEBC_DHCP4_OPTION_ARCH* PXEBC_DHCP4_OPTION_ENTRY::Arch

Referenced by PxeBcBuildDhcp4Options().

PXEBC_DHCP4_OPTION_CLID* PXEBC_DHCP4_OPTION_ENTRY::Clid

Referenced by PxeBcBuildDhcp4Options().

PXEBC_DHCP4_OPTION_MAX_MESG_SIZE* PXEBC_DHCP4_OPTION_ENTRY::MaxMesgSize

Referenced by PxeBcBuildDhcp4Options().

PXEBC_DHCP4_OPTION_MESG* PXEBC_DHCP4_OPTION_ENTRY::Mesg

Referenced by PxeBcBuildDhcp4Options().

PXEBC_DHCP4_OPTION_PARA* PXEBC_DHCP4_OPTION_ENTRY::Para

Referenced by PxeBcBuildDhcp4Options().

PXEBC_DHCP4_OPTION_UNDI* PXEBC_DHCP4_OPTION_ENTRY::Undi

Referenced by PxeBcBuildDhcp4Options().

PXEBC_DHCP4_OPTION_UUID* PXEBC_DHCP4_OPTION_ENTRY::Uuid

Referenced by PxeBcBuildDhcp4Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

PXEBC_DHCP4_OPTION_MAX_MESG_SIZE Struct Reference

Data Fields

UINT16 Size

Field Documentation

UINT16 PXEBC_DHCP4_OPTION_MAX_MESG_SIZE::Size

Referenced by PxeBcBuildDhcp4Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

PXEBC_DHCP4_OPTION_MESG Struct Reference

Data Fields

UINT8 Type

Field Documentation

UINT8 PXEBC_DHCP4_OPTION_MESG::Type

Referenced by PxeBcBuildDhcp4Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

PXEBC_DHCP4_OPTION_PARA Struct Reference

Data Fields

UINT8 ParaList [135]

Field Documentation

UINT8 PXEBC_DHCP4_OPTION_PARA::ParaList[135]

Referenced by PxeBcBuildDhcp4Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

PXEBC_DHCP4_OPTION_UNDI Struct Reference

Data Fields

UINT8 Type
UINT8 MajorVer
UINT8 MinorVer

Field Documentation

UINT8 PXEBC_DHCP4_OPTION_UNDI::MajorVer

Referenced by PxeBcBuildDhcp4Options().

UINT8 PXEBC_DHCP4_OPTION_UNDI::MinorVer

Referenced by PxeBcBuildDhcp4Options().

UINT8 PXEBC_DHCP4_OPTION_UNDI::Type

Referenced by PxeBcBuildDhcp4Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

PXEBC_DHCP4_OPTION_UUID Struct Reference

Data Fields

UINT8 Type
UINT8 Guid [16]

Field Documentation

UINT8 PXEBC_DHCP4_OPTION_UUID::Guid[16]

Referenced by PxeBcBuildDhcp4Options().

UINT8 PXEBC_DHCP4_OPTION_UUID::Type

Referenced by PxeBcBuildDhcp4Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

PXEBC_DHCP4_PACKET Union Reference

Data Fields

EFI_DHCP4_PACKET Offer
EFI_DHCP4_PACKET Ack
UINT8 Buffer [PXEBC_DHCP4_PACKET_MAX_SIZE]

Field Documentation

EFI_DHCP4_PACKET PXEBC_DHCP4_PACKET::Ack

Referenced by EfiPxeBcDiscover(), and EfiPxeBcStart().

UINT8 PXEBC_DHCP4_PACKET::Buffer[PXEBC_DHCP4_PACKET_MAX_SIZE]

EFI_DHCP4_PACKET PXEBC_DHCP4_PACKET::Offer

Referenced by EfiPxeBcStart(), PxeBcCacheDhcp4Offer(), PxeBcDhcp4BootInfo(), PxeBcDhcp4CallBack(), PxeBcHandleDhcp4Offer(), and PxeBcRetryBinlOffer().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

PXEBC_DHCP4_PACKET_CACHE Struct Reference

Data Fields

PXEBC_DHCP4_PACKET Packet
PXEBC_OFFER_TYPE OfferType
EFI_DHCP4_PACKET_OPTION * OptList [PXEBC_DHCP4_TAG_INDEX_MAX]
PXEBC_VENDOR_OPTION VendorOpt

Field Documentation

PXEBC_OFFER_TYPE PXEBC_DHCP4_PACKET_CACHE::OfferType

Referenced by PxeBcCacheDhcp4Offer(), PxeBcHandleDhcp4Offer(), PxeBcRetryBinlOffer(), PxeBcSelectBootMenu(), and PxeBcSelectBootPrompt().

EFI_DHCP4_PACKET_OPTION*
PXEBC_DHCP4_PACKET_CACHE::OptList[PXEBC_DHCP4_TAG_INDEX_MAX]

Referenced by PxeBcDhcp4BootInfo(), PxeBcHandleDhcp4Offer(), PxeBcRetryBinlOffer(), and PxeBcSelectBootPrompt().

PXEBC_DHCP4_PACKET PXEBC_DHCP4_PACKET_CACHE::Packet

Referenced by EfiPxeBcDiscover(), EfiPxeBcStart(), PxeBcCacheDhcp4Offer(), PxeBcDhcp4BootInfo(), PxeBcDhcp4CallBack(), PxeBcHandleDhcp4Offer(), and PxeBcRetryBinlOffer().

PXEBC_VENDOR_OPTION PXEBC_DHCP4_PACKET_CACHE::VendorOpt

Referenced by PxeBcDhcp4BootInfo(), PxeBcExtractDiscoverInfo(), PxeBcSelectBootMenu(), and PxeBcSelectBootPrompt().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

PXEBC_DHCP6_OPTION_ARCH Struct Reference

Data Fields

UINT16 Type

Field Documentation

UINT16 PXEBC_DHCP6_OPTION_ARCH::Type

Referenced by PxeBcBuildDhcp6Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

PXEBC_DHCP6_OPTION_ENTRY Union Reference

Data Fields

PXEBC_DHCP6_OPTION_ORO * Oro
PXEBC_DHCP6_OPTION_UNDI * Undi
PXEBC_DHCP6_OPTION_ARCH * Arch
PXEBC_DHCP6_OPTION_VENDOR_CLASS * VendorClass

Field Documentation

PXEBC_DHCP6_OPTION_ARCH* PXEBC_DHCP6_OPTION_ENTRY::Arch

Referenced by PxeBcBuildDhcp6Options().

PXEBC_DHCP6_OPTION_ORO* PXEBC_DHCP6_OPTION_ENTRY::Oro

Referenced by PxeBcBuildDhcp6Options().

PXEBC_DHCP6_OPTION_UNDI* PXEBC_DHCP6_OPTION_ENTRY::Undi

Referenced by PxeBcBuildDhcp6Options().

PXEBC_DHCP6_OPTION_VENDOR_CLASS* PXEBC_DHCP6_OPTION_ENTRY::VendorClass

Referenced by PxeBcBuildDhcp6Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

PXEBC_DHCP6_OPTION_NODE Struct Reference

Data Fields

LIST_ENTRY	Link
EFI_DHCP6_PACKET_OPTION *	Option
UINT8	Precedence

Field Documentation

LIST_ENTRY PXEBC_DHCP6_OPTION_NODE::Link

EFI_DHCP6_PACKET_OPTION* PXEBC_DHCP6_OPTION_NODE::Option

UINT8 PXEBC_DHCP6_OPTION_NODE::Precedence

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

PXEBC_DHCP6_OPTION_ORO Struct Reference

Data Fields

UINT16 OpCode [256]

Field Documentation

UINT16 PXEBC_DHCP6_OPTION_ORO::OpCode[256]

Referenced by PxeBcBuildDhcp6Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

PXEBC_DHCP6_OPTION_UNDI Struct Reference

Data Fields

UINT8 Type
UINT8 MajorVer
UINT8 MinorVer

Field Documentation

UINT8 PXEBC_DHCP6_OPTION_UNDI::MajorVer

Referenced by PxeBcBuildDhcp6Options().

UINT8 PXEBC_DHCP6_OPTION_UNDI::MinorVer

Referenced by PxeBcBuildDhcp6Options().

UINT8 PXEBC_DHCP6_OPTION_UNDI::Type

Referenced by PxeBcBuildDhcp6Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

PXEBC_DHCP6_OPTION_VENDOR_CLASS Struct Reference

Data Fields

UINT32 Vendor
UINT16 ClassLen
PXEBC_CLASS_ID ClassId

Field Documentation

PXEBC_CLASS_ID PXEBC_DHCP6_OPTION_VENDOR_CLASS::ClassId

Referenced by PxeBcBuildDhcp6Options().

UINT16 PXEBC_DHCP6_OPTION_VENDOR_CLASS::ClassLen

Referenced by PxeBcBuildDhcp6Options().

UINT32 PXEBC_DHCP6_OPTION_VENDOR_CLASS::Vendor

Referenced by PxeBcBuildDhcp6Options().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

PXEBC_DHCP6_PACKET Union Reference

Data Fields

EFI_DHCP6_PACKET Offer
EFI_DHCP6_PACKET Ack
UINT8 Buffer [PXEBC_DHCP6_PACKET_MAX_SIZE]

Field Documentation

EFI_DHCP6_PACKET PXEBC_DHCP6_PACKET::Ack

Referenced by EfiPxeBcDiscover(), and EfiPxeBcStart().

UINT8 PXEBC_DHCP6_PACKET::Buffer[PXEBC_DHCP6_PACKET_MAX_SIZE]

EFI_DHCP6_PACKET PXEBC_DHCP6_PACKET::Offer

Referenced by EfiPxeBcStart(), PxeBcCacheDhcp6Offer(), PxeBcDhcp6CallBack(), and PxeBcRetryDhcp6Binl().

Generated on Mon Sep 28 08:49:58 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

PXEBC_DHCP6_PACKET_CACHE Struct Reference

Data Fields

PXEBC_DHCP6_PACKET Packet
PXEBC_OFFER_TYPE OfferType
EFI_DHCP6_PACKET_OPTION * OptList [PXEBC_DHCP6_IDX_MAX]

Field Documentation

PXEBC_OFFER_TYPE PXEBC_DHCP6_PACKET_CACHE::OfferType

Referenced by PxeBcCacheDhcp6Offer(), PxeBcHandleDhcp6Offer(), PxeBcRetryDhcp6Binl(), PxeBcSelectBootMenu(), and PxeBcSelectBootPrompt().

EFI_DHCP6_PACKET_OPTION*
PXEBC_DHCP6_PACKET_CACHE::OptList[PXEBC_DHCP6_IDX_MAX]

Referenced by PxeBcDhcp6BootInfo(), PxeBcHandleDhcp6Offer(), and PxeBcRetryDhcp6Binl().

PXEBC_DHCP6_PACKET PXEBC_DHCP6_PACKET_CACHE::Packet

Referenced by EfiPxeBcDiscover(), EfiPxeBcStart(), PxeBcCacheDhcp6Offer(), PxeBcDhcp6CallBack(), and PxeBcRetryDhcp6Binl().

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

PXEBC_DHCP_PACKET_CACHE Union Reference

Data Fields

PXEBC_DHCP4_PACKET_CACHE Dhcp4
PXEBC_DHCP6_PACKET_CACHE Dhcp6

Field Documentation

PXEBC_DHCP4_PACKET_CACHE PXEBC_DHCP_PACKET_CACHE::Dhcp4

Referenced by EfiPxeBcDiscover(), EfiPxeBcStart(), PxeBcDhcp4CallBack(), PxeBcSelectBootMenu(), and PxeBcSelectBootPrompt().

PXEBC_DHCP6_PACKET_CACHE PXEBC_DHCP_PACKET_CACHE::Dhcp6

Referenced by EfiPxeBcDiscover(), EfiPxeBcStart(), PxeBcDhcp6CallBack(), PxeBcSelectBootMenu(), and PxeBcSelectBootPrompt().

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

PXEBC_MENU_PROMPT Struct Reference

Data Fields

UINT8 Timeout
UINT8 Prompt [1]

Field Documentation

UINT8 PXEBC_MENU_PROMPT::Prompt[1]

Referenced by PxeBcSelectBootPrompt().

UINT8 PXEBC_MENU_PROMPT::Timeout

Referenced by PxeBcSelectBootPrompt().

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

PXEBC_OPTION_BOOT_ITEM Struct Reference

Data Fields

UINT16 Type
UINT16 Layer

Field Documentation

UINT16 PXEBC_OPTION_BOOT_ITEM::Layer

Referenced by PxeBcDhcp4Discover().

UINT16 PXEBC_OPTION_BOOT_ITEM::Type

Referenced by PxeBcDhcp4Discover().

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

PXEBC_VENDOR_OPTION Struct Reference

Data Fields

UINT32	BitMap [8]
EFI_IPv4_ADDRESS	MtftpIp
UINT16	MtftpCPort
UINT16	MtftpSPort
UINT8	MtftpTimeout
UINT8	MtftpDelay
UINT8	DiscoverCtrl
EFI_IPv4_ADDRESS	DiscoverMcastIp
EFI_IPv4_ADDRESS	McastIpBase
UINT16	McastIpBlock
UINT16	McastIpRange
UINT16	BootSrvType
UINT16	BootSrvLayer
PXEBC_BOOT_SVR_ENTRY *	BootSvr
UINT8	BootSvrLen
PXEBC_BOOT_MENU_ENTRY *	BootMenu
UINT8	BootMenuLen
PXEBC_MENU_PROMPT *	MenuPrompt
UINT8	MenuPromptLen
UINT32 *	CredType
UINT8	CredTypeLen

Field Documentation

UINT32 PXEBC_VENDOR_OPTION::BitMap[8]

Referenced by PxeBcDhcp4BootInfo(), PxeBcExtractDiscoverInfo(), PxeBcSelectBootMenu(), and PxeBcSelectBootPrompt().

PXEBC_BOOT_MENU_ENTRY* PXEBC_VENDOR_OPTION::BootMenu

Referenced by PxeBcSelectBootMenu().

UINT8 PXEBC_VENDOR_OPTION::BootMenuLen

Referenced by PxeBcSelectBootMenu().

UINT16 PXEBC_VENDOR_OPTION::BootSrvLayer

UINT16 PXEBC_VENDOR_OPTION::BootSrvType

PXEBC_BOOT_SVR_ENTRY* PXEBC_VENDOR_OPTION::BootSvr

Referenced by PxeBcDhcp4BootInfo(), and PxeBcExtractDiscoverInfo().

UINT8 PXEBC_VENDOR_OPTION::BootSvrLen

Referenced by PxeBcDhcp4BootInfo(), and PxeBcExtractDiscoverInfo().

UINT32* PXEBC_VENDOR_OPTION::CredType

UINT8 PXEBC_VENDOR_OPTION::CredTypeLen

UINT8 PXEBC_VENDOR_OPTION::DiscoverCtrl

Referenced by PxeBcDhcp4BootInfo(), PxeBcExtractDiscoverInfo(), and PxeBcSelectBootPrompt().

EFI_IPv4_ADDRESS PXEBC_VENDOR_OPTION::DiscoverMcastIp

Referenced by PxeBcExtractDiscoverInfo().

EFI_IPv4_ADDRESS PXEBC_VENDOR_OPTION::McastIpBase

UINT16 PXEBC_VENDOR_OPTION::McastIpBlock

UINT16 PXEBC_VENDOR_OPTION::McastIpRange

PXEBC_MENU_PROMPT* PXEBC_VENDOR_OPTION::MenuPrompt

Referenced by PxeBcSelectBootPrompt().

UINT8 PXEBC_VENDOR_OPTION::MenuPromptLen

Referenced by PxeBcSelectBootPrompt().

UINT16 PXEBC_VENDOR_OPTION::MtftpCPort

UINT8 PXEBC_VENDOR_OPTION::MtftpDelay

EFI_IPv4_ADDRESS PXEBC_VENDOR_OPTION::MtftpIp

UINT16 PXEBC_VENDOR_OPTION::MtftpSPort

UINT8 PXEBC_VENDOR_OPTION::MtftpTimeout

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

SA_KEYMATS Struct Reference

Data Fields

EFI_IPSEC_ALGO_INFO LocalPeerInfo
EFI_IPSEC_ALGO_INFO RemotePeerInfo

Field Documentation

EFI_IPSEC_ALGO_INFO SA_KEYMATS::LocalPeerInfo

EFI_IPSEC_ALGO_INFO SA_KEYMATS::RemotePeerInfo

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

SAD_ENTRY_INDEXER Struct Reference

Data Fields

EFI_IPSEC_SA_ID SaId
UINTN Index

Field Documentation

UINTN SAD_ENTRY_INDEXER::Index

EFI_IPSEC_SA_ID SAD_ENTRY_INDEXER::SaId

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

SHELL_PARAM_PACKAGE Struct Reference

Data Fields

LIST_ENTRY Link
CHAR16 * Name
SHELL_PARAM_TYPE Type
CHAR16 * Value
UINTN OriginalPosition

Field Documentation

LIST_ENTRY SHELL_PARAM_PACKAGE::Link

CHAR16* SHELL_PARAM_PACKAGE::Name

UINTN SHELL_PARAM_PACKAGE::OriginalPosition

SHELL_PARAM_TYPE SHELL_PARAM_PACKAGE::Type

CHAR16* SHELL_PARAM_PACKAGE::Value

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

SOCK_IO_DATA Union Reference

Data Fields

VOID * RxData
VOID * TxData

Field Documentation

VOID* SOCK_IO_DATA::RxData

VOID* SOCK_IO_DATA::TxData

Referenced by SockProcessSndToken(), and SockSend().

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

SPD_ENTRY_INDEXER Struct Reference

Data Fields

UINT8 * Name
UINTN Index

Field Documentation

UINTN SPD_ENTRY_INDEXER::Index

UINT8* SPD_ENTRY_INDEXER::Name

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

STR2INT Struct Reference

Data Fields

CHAR16 * String
UINT32 Integer

Field Documentation

UINT32 STR2INT::Integer

Referenced by MapIntegerToString(), and MapStringToInteger().

CHAR16* STR2INT::String

Referenced by MapIntegerToString(), and MapStringToInteger().

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

TCP_ACCESS_POINT Union Reference

Data Fields

EFI_TCP4_ACCESS_POINT Tcp4Ap
EFI_TCP6_ACCESS_POINT Tcp6Ap

Field Documentation

EFI_TCP4_ACCESS_POINT TCP_ACCESS_POINT::Tcp4Ap

EFI_TCP6_ACCESS_POINT TCP_ACCESS_POINT::Tcp6Ap

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

TCP_CONFIG_DATA Union Reference

Data Fields

EFI_TCP4_CONFIG_DATA Tcp4CfgData
EFI_TCP6_CONFIG_DATA Tcp6CfgData

Field Documentation

EFI_TCP4_CONFIG_DATA TCP_CONFIG_DATA::Tcp4CfgData

EFI_TCP6_CONFIG_DATA TCP_CONFIG_DATA::Tcp6CfgData

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

TCP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT Struct Reference

Data Fields

EFI_SERVICE_BINDING_PROTOCOL * ServiceBinding
UINTN NumberOfChildren
EFI_HANDLE * ChildHandleBuffer

Field Documentation

EFI_HANDLE* TCP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer

Referenced by TcpDestroyService().

UINTN TCP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren

Referenced by TcpDestroyService().

EFI_SERVICE_BINDING_PROTOCOL*
TCP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding

Referenced by TcpDestroyService().

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

TRAFFIC_SELECTOR Struct Reference

Data Fields

UINT8 TSType
UINT8 IpProtocolId
UINT16 SelecorLen
UINT16 StartPort
UINT16 EndPort

Field Documentation

UINT16 TRAFFIC_SELECTOR::EndPort

Referenced by Ikev2DecodePayload(), Ikev2EncodePayload(), and Ikev2GenerateTsPayload().

UINT8 TRAFFIC_SELECTOR::IpProtocolId

Referenced by Ikev2GenerateTsPayload().

UINT16 TRAFFIC_SELECTOR::SelecorLen

Referenced by Ikev2DecodePayload(), Ikev2EncodePayload(), and Ikev2GenerateTsPayload().

UINT16 TRAFFIC_SELECTOR::StartPort

Referenced by Ikev2DecodePayload(), Ikev2EncodePayload(), and Ikev2GenerateTsPayload().

UINT8 TRAFFIC_SELECTOR::TSType

Referenced by Ikev2GenerateTsPayload().

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

UDP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT Struct Reference

Data Fields

EFI_SERVICE_BINDING_PROTOCOL * ServiceBinding
UINTN NumberOfChildren
EFI_HANDLE * ChildHandleBuffer

Field Documentation

EFI_HANDLE* UDP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer

Referenced by Udp6DriverBindingStop().

UINTN UDP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren

Referenced by Udp6DriverBindingStop().

EFI_SERVICE_BINDING_PROTOCOL*
UDP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding

Referenced by Udp6DriverBindingStop().

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- Data Structures
- Data Fields

VAR_CHECK_ITEM Struct Reference

Data Fields

CHAR16 * VarName
UINT32 Attribute1
UINT32 Attribute2
UINT32 Attribute3
UINT32 Attribute4

Field Documentation

UINT32 VAR_CHECK_ITEM::Attribute1

Referenced by IpSecConfigRetriveCheckListByName().

UINT32 VAR_CHECK_ITEM::Attribute2

Referenced by IpSecConfigRetriveCheckListByName().

UINT32 VAR_CHECK_ITEM::Attribute3

Referenced by IpSecConfigRetriveCheckListByName().

UINT32 VAR_CHECK_ITEM::Attribute4

Referenced by IpSecConfigRetriveCheckListByName().

CHAR16* VAR_CHECK_ITEM::VarName

Referenced by IpSecConfigRetriveCheckListByName().

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

- c
- d
- e
- f
- g
- h
- i
- k
- l
- m
- n
- o
- p
- q
- r
- s
- t
- u
- v
- w
- x
- z

Here is a list of all struct and union fields with links to the structures/unions they belong to:

- a -

- AA : _DNS_FLAGS
- Ack : HTTP_BOOT_DHCP4_PACKET , PXEBC_DHCP6_PACKET , PXEBC_DHCP4_PACKET , _TCP_SEG
- Action : _IPSEC_SPD_DATA
- ActiveR2Ts : _ISCSI_TCB
- Address : HTTP_BOOT_URI_CONTENT , IP6_MLD_GROUP , _IP6_ADDRESS_INFO , IP6_DESTROY_CHILD_BY_ADDR_CALLBACK_CONTEXT
- AddressCount : _IP6_INTERFACE
- AddressInfo : _IP6_DAD_ENTRY , _IP6_DELAY_JOIN_LIST
- AddressList : _IP6_INTERFACE
- AddrInfo : IP6_ADDRESS_INFO_ENTRY
- AddrIsOk : HTTP_BOOT_URI_CONTENT
- AddrLink : _IP6_PROTOCOL
- AditionalNum : DNS_HEADER
- AdPref : _DHCP6_INSTANCE
- AdSelect : _DHCP6_INSTANCE
- AfterEncodePayload : IKEV2_SESSION_COMMON
- AlgGuid : IKE_ALG_GUID_INFO
- AlgoInfo : _IPSEC_SAD_DATA
- AlgorithmId : _ENCRYPT_ALGORITHM , _AUTH_ALGORITHM , _HASH_ALGORITHM , IKE_ALG_GUID_INFO
- AlgSize : IKE_ALG_GUID_INFO
- AllCacheLink : DNS4_CACHE , DNS6_CACHE
- AllExpireTime : _DHCP6_IA_CB
- AllServerLink : DNS4_SERVER_IP , DNS6_SERVER_IP
- AltIfId : IP6_CONFIG_DATA , _IP6_CONFIG_INSTANCE
- AnswersNum : DNS_HEADER

- AntiReplayBitmap : _IPSEC_SAD_DATA
- AntiReplayWindowSize : _IPSEC_SAD_DATA
- Arch : HTTP_BOOT_DHCP4_OPTION_ENTRY , PXEBC_DHCP6_OPTION_ENTRY , PXEBC_DHCP4_OPTION_ENTRY
- ArchitecturePrefix : HTTP_BOOT_DHCP4_OPTION_CLID , PXEBC_CLASS_ID , PXEBC_DHCP4_OPTION_CLID
- ArchitectureType : HTTP_BOOT_DHCP4_OPTION_CLID , PXEBC_CLASS_ID , PXEBC_DHCP4_OPTION_CLID
- Arg : _ARG_LIST
- Arp : _PXEBC_PRIVATE_DATA
- ArpChild : _PXEBC_PRIVATE_DATA
- ArpFree : _IP6_NEIGHBOR_ENTRY
- ArpList : _IP6_NEIGHBOR_ENTRY
- ArpQues : _IP6_INTERFACE
- ArpUpdateEvent : _PXEBC_PRIVATE_DATA
- Assemble : _IP6_SERVICE
- AsyncEvent : _ISCSI_ASYNC_MESSAGE
- AsyncVCode : _ISCSI_ASYNC_MESSAGE
- AttemptConfigIndex : _ISCSI_ATTEMPT_CONFIG_NVDATA
- AttemptConfigs : ISCSI_PRIVATE_DATA
- AttemptCount : ISCSI_PRIVATE_DATA
- AttemptName : _ISCSI_ATTEMPT_CONFIG_NVDATA , _ISCSI_CONFIG_IFR_NVDATA
- AttemptTitleHelpToken : _ISCSI_ATTEMPT_CONFIG_NVDATA
- AttemptTitleToken : _ISCSI_ATTEMPT_CONFIG_NVDATA
- Attr : IKE_SA_ATTRIBUTE
- Attribute : IKEV2_TRANSFORM_DATA , IP6_CONFIG_DATA_ITEM
- Attribute1 : VAR_CHECK_ITEM
- Attribute2 : VAR_CHECK_ITEM
- Attribute3 : VAR_CHECK_ITEM
- Attribute4 : VAR_CHECK_ITEM
- AttritType : IKEV2_CFG_ATTRIBUTES
- AttrLength : IKE_SA_ATTR_UNION
- AttrType : IKE_SA_ATTRIBUTE
- AttrValue : IKE_SA_ATTR_UNION
- AuthConfig : _ISCSI_CHAP_AUTH_DATA
- AuthConfigData : _ISCSI_ATTEMPT_CONFIG_NVDATA
- AuthData : _ISCSI_SESSION
- AuthenticationType : _ISCSI_CONFIG_IFR_NVDATA , _ISCSI_ATTEMPT_CONFIG_NVDATA
- AuthMethod : IKEV2_AUTH
- Authority : HTTP_BOOT_URI_CONTENT
- AuthorityNum : DNS_HEADER
- AuthStep : _ISCSI_CONNECTION
- AuthType : _ISCSI_SESSION
- AutoConfigureMode : _ISCSI_ATTEMPT_CONFIG_NVDATA
- AutoConfigureSuccess : _ISCSI_ATTEMPT_CONFIG_NVDATA
- AutonomousPrefix : _IP6_SERVICE

File List

Here is a list of all files with brief descriptions:

NetworkPkg/NetworkPkg.dec.decdoxygen
NetworkPkg/Application/IfConfig6/IfConfig6.c
NetworkPkg/Application/IfConfig6/IfConfig6.h
NetworkPkg/Application/IpsecConfig/Delete.c
NetworkPkg/Application/IpsecConfig/Delete.h
NetworkPkg/Application/IpsecConfig/Dump.c
NetworkPkg/Application/IpsecConfig/Dump.h
NetworkPkg/Application/IpsecConfig/ForEach.c
NetworkPkg/Application/IpsecConfig/ForEach.h
NetworkPkg/Application/IpsecConfig/Helper.c
NetworkPkg/Application/IpsecConfig/Helper.h
NetworkPkg/Application/IpsecConfig/Indexer.c
NetworkPkg/Application/IpsecConfig/Indexer.h
NetworkPkg/Application/IpsecConfig/IpSecConfig.c
NetworkPkg/Application/IpsecConfig/IpSecConfig.h
NetworkPkg/Application/IpsecConfig/Match.c
NetworkPkg/Application/IpsecConfig/Match.h
NetworkPkg/Application/IpsecConfig/PolicyEntryOperation.c
NetworkPkg/Application/IpsecConfig/PolicyEntryOperation.h
NetworkPkg/Application/Ping6/Ping6.c
NetworkPkg/Application/Ping6/Ping6.h
NetworkPkg/Application/Ping6/Ia32/Tsc.c
NetworkPkg/Application/Ping6/Ipf/Itc.c
NetworkPkg/Application/Ping6/X64/Tsc.c
NetworkPkg/Application/VConfig/VConfig.c
NetworkPkg/Dhcp6Dxe/ComponentName.c
NetworkPkg/Dhcp6Dxe/Dhcp6Driver.c
NetworkPkg/Dhcp6Dxe/Dhcp6Driver.h
NetworkPkg/Dhcp6Dxe/Dhcp6Impl.c
NetworkPkg/Dhcp6Dxe/Dhcp6Impl.h
NetworkPkg/Dhcp6Dxe/Dhcp6Io.c
NetworkPkg/Dhcp6Dxe/Dhcp6Io.h
NetworkPkg/Dhcp6Dxe/Dhcp6Utility.c
NetworkPkg/Dhcp6Dxe/Dhcp6Utility.h
NetworkPkg/DnsDxe/ComponentName.c
NetworkPkg/DnsDxe/DnsDhcp.c
NetworkPkg/DnsDxe/DnsDhcp.h
NetworkPkg/DnsDxe/DnsDriver.c
NetworkPkg/DnsDxe/DnsDriver.h
NetworkPkg/DnsDxe/DnsImpl.c
NetworkPkg/DnsDxe/DnsImpl.h

NetworkPkg/DnsDxe/DnsProtocol.c
NetworkPkg/HttpBootDxe/HttpBootClient.c
NetworkPkg/HttpBootDxe/HttpBootClient.h
NetworkPkg/HttpBootDxe/HttpBootComponentName.c
NetworkPkg/HttpBootDxe/HttpBootComponentName.h
NetworkPkg/HttpBootDxe/HttpBootDhcp4.c
NetworkPkg/HttpBootDxe/HttpBootDhcp4.h
NetworkPkg/HttpBootDxe/HttpBootDxe.c
NetworkPkg/HttpBootDxe/HttpBootDxe.h
NetworkPkg/HttpBootDxe/HttpBootImpl.c
NetworkPkg/HttpBootDxe/HttpBootImpl.h
NetworkPkg/HttpBootDxe/HttpBootSupport.c
NetworkPkg/HttpBootDxe/HttpBootSupport.h
NetworkPkg/HttpDxe/ComponentName.c
NetworkPkg/HttpDxe/ComponentName.h
NetworkPkg/HttpDxe/HttpDns.c
NetworkPkg/HttpDxe/HttpDns.h
NetworkPkg/HttpDxe/HttpDriver.c
NetworkPkg/HttpDxe/HttpDriver.h
NetworkPkg/HttpDxe/HttpImpl.c
NetworkPkg/HttpDxe/HttpImpl.h
NetworkPkg/HttpDxe/HttpProto.c
NetworkPkg/HttpDxe/HttpProto.h
NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesDxe.c
NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesDxe.h
NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesImpl.c
NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesProtocol.c
NetworkPkg/Include/Guid/Ip6ConfigHii.h
NetworkPkg/Include/Guid/IScsiConfigHii.h
NetworkPkg/Ip6Dxe/ComponentName.c
NetworkPkg/Ip6Dxe/Ip6Common.c
NetworkPkg/Ip6Dxe/Ip6Common.h
NetworkPkg/Ip6Dxe/Ip6Config.vfr
NetworkPkg/Ip6Dxe/Ip6ConfigImpl.c
NetworkPkg/Ip6Dxe/Ip6ConfigImpl.h
NetworkPkg/Ip6Dxe/Ip6ConfigNv.c
NetworkPkg/Ip6Dxe/Ip6ConfigNv.h
NetworkPkg/Ip6Dxe/Ip6Driver.c
NetworkPkg/Ip6Dxe/Ip6Driver.h
NetworkPkg/Ip6Dxe/Ip6Icmp.c
NetworkPkg/Ip6Dxe/Ip6Icmp.h
NetworkPkg/Ip6Dxe/Ip6If.c
NetworkPkg/Ip6Dxe/Ip6If.h
NetworkPkg/Ip6Dxe/Ip6Impl.c
NetworkPkg/Ip6Dxe/Ip6Impl.h

NetworkPkg/Ip6Dxe/Ip6Input.c
NetworkPkg/Ip6Dxe/Ip6Input.h
NetworkPkg/Ip6Dxe/Ip6Mld.c
NetworkPkg/Ip6Dxe/Ip6Mld.h
NetworkPkg/Ip6Dxe/Ip6Nd.c
NetworkPkg/Ip6Dxe/Ip6Nd.h
NetworkPkg/Ip6Dxe/Ip6NvData.h
NetworkPkg/Ip6Dxe/Ip6Option.c
NetworkPkg/Ip6Dxe/Ip6Option.h
NetworkPkg/Ip6Dxe/Ip6Output.c
NetworkPkg/Ip6Dxe/Ip6Output.h
NetworkPkg/Ip6Dxe/Ip6Route.c
NetworkPkg/Ip6Dxe/Ip6Route.h
NetworkPkg/IpSecDxe/ComponentName.c
NetworkPkg/IpSecDxe/IetfConstants.c
NetworkPkg/IpSecDxe/Ike.h
NetworkPkg/IpSecDxe/IkeCommon.c
NetworkPkg/IpSecDxe/IkeCommon.h
NetworkPkg/IpSecDxe/IkePacket.c
NetworkPkg/IpSecDxe/IkePacket.h
NetworkPkg/IpSecDxe/IkeService.c
NetworkPkg/IpSecDxe/IkeService.h
NetworkPkg/IpSecDxe/IpSecConfigImpl.c
NetworkPkg/IpSecDxe/IpSecConfigImpl.h
NetworkPkg/IpSecDxe/IpSecCryptIo.c
NetworkPkg/IpSecDxe/IpSecCryptIo.h
NetworkPkg/IpSecDxe/IpSecDebug.c
NetworkPkg/IpSecDxe/IpSecDebug.h
NetworkPkg/IpSecDxe/IpSecDriver.c
NetworkPkg/IpSecDxe/IpSecImpl.c
NetworkPkg/IpSecDxe/IpSecImpl.h
NetworkPkg/IpSecDxe/IpSecMain.c
NetworkPkg/IpSecDxe/Ikev2/ChildSa.c
NetworkPkg/IpSecDxe/Ikev2/Exchange.c
NetworkPkg/IpSecDxe/Ikev2/Ikev2.h
NetworkPkg/IpSecDxe/Ikev2/Info.c
NetworkPkg/IpSecDxe/Ikev2/Payload.c
NetworkPkg/IpSecDxe/Ikev2/Payload.h
NetworkPkg/IpSecDxe/Ikev2/Sa.c
NetworkPkg/IpSecDxe/Ikev2/Utility.c
NetworkPkg/IpSecDxe/Ikev2/Utility.h
NetworkPkg/IScsiDxe/ComponentName.c
NetworkPkg/IScsiDxe/IScsiAuthenticationInfo.c
NetworkPkg/IScsiDxe/IScsiCHAP.c
NetworkPkg/IScsiDxe/IScsiCHAP.h

NetworkPkg/IScsiDxe/IScsiConfig.c
NetworkPkg/IScsiDxe/IScsiConfig.h
NetworkPkg/IScsiDxe/IScsiConfigNVDataStruc.h
NetworkPkg/IScsiDxe/IScsiConfigVfr.vfr
NetworkPkg/IScsiDxe/IScsiDhcp.c
NetworkPkg/IScsiDxe/IScsiDhcp.h
NetworkPkg/IScsiDxe/IScsiDhcp6.c
NetworkPkg/IScsiDxe/IScsiDhcp6.h
NetworkPkg/IScsiDxe/IScsiDriver.c
NetworkPkg/IScsiDxe/IScsiDriver.h
NetworkPkg/IScsiDxe/IScsiExtScsiPassThru.c
NetworkPkg/IScsiDxe/IScsiIbft.c
NetworkPkg/IScsiDxe/IScsiIbft.h
NetworkPkg/IScsiDxe/IScsiImpl.h
NetworkPkg/IScsiDxe/IScsiInitiatorName.c
NetworkPkg/IScsiDxe/IScsiMisc.c
NetworkPkg/IScsiDxe/IScsiMisc.h
NetworkPkg/IScsiDxe/IScsiProto.c
NetworkPkg/IScsiDxe/IScsiProto.h
NetworkPkg/Mtftp6Dxe/ComponentName.c
NetworkPkg/Mtftp6Dxe/Mtftp6Driver.c
NetworkPkg/Mtftp6Dxe/Mtftp6Driver.h
NetworkPkg/Mtftp6Dxe/Mtftp6Impl.c
NetworkPkg/Mtftp6Dxe/Mtftp6Impl.h
NetworkPkg/Mtftp6Dxe/Mtftp6Option.c
NetworkPkg/Mtftp6Dxe/Mtftp6Option.h
NetworkPkg/Mtftp6Dxe/Mtftp6Rrq.c
NetworkPkg/Mtftp6Dxe/Mtftp6Support.c
NetworkPkg/Mtftp6Dxe/Mtftp6Support.h
NetworkPkg/Mtftp6Dxe/Mtftp6Wrq.c
NetworkPkg/TcpDxe/ComponentName.c
NetworkPkg/TcpDxe/Socket.h
NetworkPkg/TcpDxe/SockImpl.c
NetworkPkg/TcpDxe/SockImpl.h
NetworkPkg/TcpDxe/SockInterface.c
NetworkPkg/TcpDxe/TcpDispatcher.c
NetworkPkg/TcpDxe/TcpDriver.c
NetworkPkg/TcpDxe/TcpDriver.h
NetworkPkg/TcpDxe/TcpFunc.h
NetworkPkg/TcpDxe/TcpInput.c
NetworkPkg/TcpDxe/TcpIo.c
NetworkPkg/TcpDxe/TcpMain.c
NetworkPkg/TcpDxe/TcpMain.h
NetworkPkg/TcpDxe/TcpMisc.c
NetworkPkg/TcpDxe/TcpOption.c

NetworkPkg/TcpDxe/TcpOption.h
NetworkPkg/TcpDxe/TcpOutput.c
NetworkPkg/TcpDxe/TcpProto.h
NetworkPkg/TcpDxe/TcpTimer.c
NetworkPkg/Udp6Dxe/ComponentName.c
NetworkPkg/Udp6Dxe/Udp6Driver.c
NetworkPkg/Udp6Dxe/Udp6Driver.h
NetworkPkg/Udp6Dxe/Udp6Impl.c
NetworkPkg/Udp6Dxe/Udp6Impl.h
NetworkPkg/Udp6Dxe/Udp6Main.c
NetworkPkg/UefiPxeBcDxe/ComponentName.c
NetworkPkg/UefiPxeBcDxe/PxeBcBoot.c
NetworkPkg/UefiPxeBcDxe/PxeBcBoot.h
NetworkPkg/UefiPxeBcDxe/PxeBcDhcp4.c
NetworkPkg/UefiPxeBcDxe/PxeBcDhcp4.h
NetworkPkg/UefiPxeBcDxe/PxeBcDhcp6.c
NetworkPkg/UefiPxeBcDxe/PxeBcDhcp6.h
NetworkPkg/UefiPxeBcDxe/PxeBcDriver.c
NetworkPkg/UefiPxeBcDxe/PxeBcDriver.h
NetworkPkg/UefiPxeBcDxe/PxeBcImpl.c
NetworkPkg/UefiPxeBcDxe/PxeBcImpl.h
NetworkPkg/UefiPxeBcDxe/PxeBcMtftp.c
NetworkPkg/UefiPxeBcDxe/PxeBcMtftp.h
NetworkPkg/UefiPxeBcDxe/PxeBcSupport.c
NetworkPkg/UefiPxeBcDxe/PxeBcSupport.h

Generated on Mon Sep 28 08:49:59 2015 for NetworkPkg[ALL] by 1.5.7.1

NetworkPkg/Application/IfConfig6/IfConfig6.c File Reference

Functions

ARG_LIST * SplitStrToList (IN CONST CHAR16 *String, IN CHAR16 Separator)
VAR_CHECK_CODE IfConfig6RetriveCheckListByName (IN VAR_CHECK_ITEM *CheckList, IN CHAR16 *Name, IN BOOLEAN Init)
VOID EFIAPI IfConfig6ManualAddressNotify (IN EFI_EVENT Event, IN VOID *Context)
VOID IfConfig6PrintMacAddr (IN UINT8 *Node, IN UINT32 Size)
VOID IfConfig6PrintIpAddr (IN EFI_IPv6_ADDRESS *Ip, IN UINT8 *PrefixLen)
EFI_STATUS IfConfig6ParseManualAddressList (IN OUT ARG_LIST **Arg, OUT EFI_IP6_CONFIG_MANUAL_ADDRESS **Buf, OUT UINTN *BufSize)
EFI_STATUS IfConfig6ParseGwDnsAddressList (IN OUT ARG_LIST **Arg, OUT EFI_IPv6_ADDRESS **Buf, OUT UINTN *BufSize)
EFI_STATUS IfConfig6ParseInterfaceId (IN OUT ARG_LIST **Arg, OUT EFI_IP6_CONFIG_INTERFACE_ID **IfId)
EFI_STATUS IfConfig6ParseDadXmits (IN OUT ARG_LIST **Arg, OUT UINT32 *Xmits)
EFI_STATUS IfConfig6GetInterfaceInfo (IN EFI_HANDLE ImageHandle, IN CHAR16 *IfName, IN LIST_ENTRY *IfList)
EFI_STATUS IfConfig6ShowInterfaceInfo (IN LIST_ENTRY *IfList)
EFI_STATUS IfConfig6ClearInterfaceInfo (IN LIST_ENTRY *IfList)
EFI_STATUS IfConfig6SetInterfaceInfo (IN LIST_ENTRY *IfList, IN ARG_LIST *VarArg)
EFI_STATUS IfConfig6 (IN IFCONFIG6_PRIVATE_DATA *Private)
VOID IfConfig6Cleanup (IN IFCONFIG6_PRIVATE_DATA *Private)
EFI_STATUS EFIAPI IfConfig6Initialize (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)

Variables

EFI_HII_HANDLE mHiiHandle
SHELL_PARAM_ITEM mIfConfig6CheckList []
VAR_CHECK_ITEM mSetCheckList []

Detailed Description

The implementation for Shell application IfConfig6.

Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS IfConfig6 (IN IFCONFIG6_PRIVATE_DATA * *Private*)

The IfConfig6 main process.

Parameters:

[in] *Private* The pointer of IFCONFIG6_PRIVATE_DATA.

Return values:

EFI_SUCCESS IfConfig6 processed successfully.
others The IfConfig6 process failed.

References IfConfig6ClearInterfaceInfo(), IfConfig6GetInterfaceInfo(), IfConfig6OpClear, IfConfig6OpList, IfConfig6OpSet, IfConfig6SetInterfaceInfo(), and IfConfig6ShowInterfaceInfo().

Referenced by IfConfig6Initialize().

VOID IfConfig6Cleanup (IN IFCONFIG6_PRIVATE_DATA * *Private*)

The IfConfig6 cleanup process, free the allocated memory.

Parameters:

[in] *Private* The pointer of IFCONFIG6_PRIVATE_DATA.

References _IFCONFIG6_INTERFACE_CB::IfId, _IFCONFIG6_INTERFACE_CB::IfInfo, _IFCONFIG6_INTERFACE_CB::Link, and _ARG_LIST::Next.

Referenced by IfConfig6Initialize().

EFI_STATUS IfConfig6ClearInterfaceInfo (IN LIST_ENTRY * *IfList*)

The clean process of the IfConfig6 application.

Parameters:

[in] *IfList* The pointer of IfList(interface list).

Return values:

EFI_SUCCESS The IfConfig6 clean processed successfully.
others The IfConfig6 clean process failed.

References _IFCONFIG6_INTERFACE_CB::IfCfg, mHiiHandle, and Policy.

Referenced by IfConfig6().

```
EFI_STATUS IfConfig6GetInterfaceInfo ( IN EFI_HANDLE     ImageHandle,
                                       IN CHAR16 *       IfName,
                                       IN LIST_ENTRY *   IfList
                                     )
```

The get current status of all handles.

Parameters:

[in]	*ImageHandle*	The handle of ImageHandle.
[in]	*IfName*	The pointer of IfName(interface name).
[in]	*IfList*	The pointer of IfList(interface list).

Return values:

EFI_SUCCESS	The get status processed successfully.
others	The get status process failed.

References _IFCONFIG6_INTERFACE_CB::DnsAddr, _IFCONFIG6_INTERFACE_CB::DnsCnt, _IFCONFIG6_INTERFACE_CB::IfCfg, _IFCONFIG6_INTERFACE_CB::IfId, _IFCONFIG6_INTERFACE_CB::IfInfo, _IFCONFIG6_INTERFACE_CB::Link, mHiiHandle, _IFCONFIG6_INTERFACE_CB::NicHandle, _IFCONFIG6_INTERFACE_CB::Policy, and _IFCONFIG6_INTERFACE_CB::Xmits.

Referenced by IfConfig6().

```
EFI_STATUS EFIAPI IfConfig6Initialize ( IN EFI_HANDLE          ImageHandle,
                                        IN EFI_SYSTEM_TABLE *  SystemTable
                                      )
```

This is the declaration of an EFI image entry point. This entry point is the same for UEFI Applications, UEFI OS Loaders, and UEFI Drivers, including both device drivers and bus drivers.

The entry point for the IfConfig6 application which parses the command line input and calls the IfConfig6 process.

Parameters:

[in]	*ImageHandle*	The image handle of this application.
[in]	*SystemTable*	The pointer to the EFI System Table.

Return values:

EFI_SUCCESS	The operation completed successfully.
Others	Some errors occur.

References _ARG_LIST::Arg, IfConfig6(), IfConfig6Cleanup(), IfConfig6OpClear, IfConfig6OpList, IfConfig6OpSet, _IFCONFIG6_PRIVATE_DATA::IfList, _IFCONFIG6_PRIVATE_DATA::IfName, mHiiHandle, mIfConfig6CheckList, _ARG_LIST::Next, _IFCONFIG6_PRIVATE_DATA::OpCode, SplitStrToList(), and _IFCONFIG6_PRIVATE_DATA::VarArg.

```
VOID EFIAPI IfConfig6ManualAddressNotify ( IN EFI_EVENT  Event,
                                           IN VOID *     Context
                                         )
```

The notify function of create event when performing a manual config.

Parameters:

[in]	*Event*	The event this notify function registered to.
[in]	*Context*	Pointer to the context data registered to the event.

Referenced by IfConfig6SetInterfaceInfo().

```
EFI_STATUS IfConfig6ParseDadXmits ( IN OUT ARG_LIST **   Arg,
                                    OUT UINT32 *         Xmits
                                  )
```

Parse dad in string format from Args with the "-s" option and convert it to UINT32 format.

Parameters:

[in,out]	*Arg*	The pointer of the address of ARG_LIST that saves Args with the "-s" option.
[out]	*Xmits*	The pointer of Xmits.

Return values:

EFI_SUCCESS	The get status processed successfully.
others	The get status process failed.

Referenced by IfConfig6SetInterfaceInfo().

```
EFI_STATUS IfConfig6ParseGwDnsAddressList ( IN OUT ARG_LIST **   Arg,
                                            OUT
                                            EFI_IPv6_ADDRESS     Buf,
                                            **
                                            OUT UINTN *          BufSize
                                          )
```

Pick up gw/dns IPv6 address in string format from Args with "-s" option and convert it to EFI_IPv6_ADDRESS format.

Parameters:

[in,out]	*Arg*	The pointer of the address of ARG_LIST that save Args with the "-s" option.
[out]	*Buf*	The pointer of the address of EFI_IPv6_ADDRESS.
[out]	*BufSize*	The pointer of BufSize that describes the size of Buf in bytes.

Return values:

EFI_SUCCESS	The conversion is successful.
Others	Doesn't find the host address, or it is an invalid IPv6 address in string format.

References _ARG_LIST::Arg, and _ARG_LIST::Next.

Referenced by IfConfig6SetInterfaceInfo().

```
EFI_STATUS IfConfig6ParseInterfaceId ( IN OUT ARG_LIST **                    Arg,
                                       OUT EFI_IP6_CONFIG_INTERFACE_ID **    IfId
                                     )
```

Parse InterfaceId in string format from Args with the "-s" option and convert it to EFI_IP6_CONFIG_INTERFACE_ID format.

Parameters:

[in,out]	*Arg*	The pointer of the address of ARG_LIST that saves Args with the "-s" option.
[out]	*IfId*	The pointer of EFI_IP6_CONFIG_INTERFACE_ID.

Return values:

EFI_SUCCESS	The get status processed successfullly.
EFI_INVALID_PARAMETER	The get status process failed.

Referenced by IfConfig6SetInterfaceInfo().

```
EFI_STATUS
IfConfig6ParseManualAddressList ( IN OUT ARG_LIST **                          Arg,
                                  OUT EFI_IP6_CONFIG_MANUAL_ADDRESS **        Buf,
                                  OUT UINTN *                                 BufSize
                                )
```

Pick up host IPv6 address in string format from Args with "-s" option and convert it to EFI_IP6_CONFIG_MANUAL_ADDRESS format.

Parameters:

[in,out]	*Arg*	The pointer of the address of ARG_LIST which save Args with the "-s" option.
[out]	*Buf*	The pointer of the address of EFI_IP6_CONFIG_MANUAL_ADDRESS.
[out]	*BufSize*	The pointer of BufSize that describes the size of Buf in bytes.

Return values:

EFI_SUCCESS	The convertion is successful.
Others	Does't find the host address, or it is an invalid IPv6 address in string format.

References _ARG_LIST::Arg, and _ARG_LIST::Next.

Referenced by IfConfig6SetInterfaceInfo().

```
VOID IfConfig6PrintIpAddr ( IN EFI_IPv6_ADDRESS *   Ip,
                            IN UINT8 *              PrefixLen
                          )
```

Print IPv6 address.

Parameters:

[in]	*Ip*	The pointer of Ip bufffer in EFI_IPv6_ADDRESS format.
[in]	*PrefixLen*	The pointer of PrefixLen that describes the size Prefix.

References mHiiHandle, and PREFIXMAXLEN.

Referenced by IfConfig6SetInterfaceInfo(), and IfConfig6ShowInterfaceInfo().

```
VOID IfConfig6PrintMacAddr ( IN UINT8 *   Node,
                             IN UINT32    Size
                           )
```

Print MAC address.

Parameters:

[in]	*Node*	The pointer of MAC address buffer.
[in]	*Size*	The size of MAC address buffer.

References MACADDRMAXSIZE, and mHiiHandle.

Referenced by IfConfig6ShowInterfaceInfo().

```
VAR_CHECK_CODE IfConfig6RetriveCheckListByName ( IN VAR_CHECK_ITEM *   CheckList,
                                                 IN CHAR16 *           Name,
                                                 IN BOOLEAN            Init
                                               )
```

Check the correctness of input Args with '-s' option.

Parameters:

[in]	*CheckList*	The pointer of VAR_CHECK_ITEM array.
[in]	*Name*	The pointer of input arg.
[in]	*Init*	The switch to execute the check.

Returns:

The value of VAR_CHECK_CODE.

References VarCheckConflict, VarCheckDuplicate, VarCheckOk, and VarCheckUnknown.

Referenced by IfConfig6SetInterfaceInfo().

```
EFI_STATUS IfConfig6SetInterfaceInfo ( IN LIST_ENTRY *   IfList,
                                       IN ARG_LIST *     VarArg
                                     )
```

The set process of the IfConfig6 application.

Parameters:

[in]	*IfList*	The pointer of IfList(interface list).
[in]	*VarArg*	The pointer of ARG_LIST(Args with "-s" option).

Return values:

EFI_SUCCESS	The IfConfig6 set processed successfully.
others	The IfConfig6 set process failed.

References _IFCONFIG6_INTERFACE_CB::IfCfg, IfConfig6ManualAddressNotify(), IfConfig6ParseDadXmits(), IfConfig6ParseGwDnsAddressList(), IfConfig6ParseInterfaceId(), IfConfig6ParseManualAddressList(), IfConfig6PrintIpAddr(), IfConfig6RetriveCheckListByName(), mHiiHandle, Policy, VarCheckConflict, VarCheckDuplicate, VarCheckOk, and VarCheckUnknown.

Referenced by IfConfig6().

```
EFI_STATUS IfConfig6ShowInterfaceInfo ( IN LIST_ENTRY *   IfList   )
```

The list process of the IfConfig6 application.

Parameters:

[in]	*IfList*	The pointer of IfList(interface list).

Return values:

EFI_SUCCESS	The IfConfig6 list processed successfully.
others	The IfConfig6 list process failed.

References _IFCONFIG6_INTERFACE_CB::DnsAddr, _IFCONFIG6_INTERFACE_CB::DnsCnt, IfConfig6PrintIpAddr(), IfConfig6PrintMacAddr(), _IFCONFIG6_INTERFACE_CB::IfId, _IFCONFIG6_INTERFACE_CB::IfInfo, mHiiHandle, _IFCONFIG6_INTERFACE_CB::Policy, and _IFCONFIG6_INTERFACE_CB::Xmits.

Referenced by IfConfig6().

ARG_LIST* SplitStrToList (IN CONST CHAR16 * *String*,
IN CHAR16 *Separator*
)

Split a string with specified separator and save the substring to a list.

Parameters:

[in]	*String*	The pointer of the input string.
[in]	*Separator*	The specified separator.

Returns:

The pointer of headnode of ARG_LIST.

References _ARG_LIST::Arg, and _ARG_LIST::Next.

Referenced by IfConfig6Initialize().

Variable Documentation

EFI_HII_HANDLE mHiiHandle

Referenced by AddOrInsertPolicyEntry(), AddVlan(), CombineSadEntry(), CombineSpdEntry(), ConstructPadIndexer(), ConstructSadIndexer(), CreatePadEntry(), CreateSadEntry(), CreateSpdEntry(), CreateSpdSelector(), DeleteVlan(), EditPolicyEntry(), FlushOrDeletePolicyEntry(), GetNumber(), IfConfig6ClearInterfaceInfo(), IfConfig6GetInterfaceInfo(), IfConfig6Initialize(), IfConfig6PrintIpAddr(), IfConfig6PrintMacAddr(), IfConfig6SetInterfaceInfo(), IfConfig6ShowInterfaceInfo(), InitializeIpSecConfig(), InitializePing6(), LocateNicHandleBuffer(), NicNameToHandle(), Ping6(), Ping6CreateIp6Instance(), Ping6OnEchoReplyReceived(), Ping6OnTimerRoutine(), ShowNicVlanInfo(), and VlanConfigMain().

SHELL_PARAM_ITEM mIfConfig6CheckList[]

Initial value:

```
{
  {
    L"-b",
    TypeFlag
  },
  {
    L"-s",
    TypeMaxValue
  },
  {
    L"-l",
    TypeValue
  },
  {
```

```
    L"-r",
    TypeValue
  },
  {
    L"-?",
    TypeFlag
  },
  {
    NULL,
    TypeMax
  },
}
```

Referenced by IfConfig6Initialize().

VAR_CHECK_ITEM mSetCheckList[]

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Application/IfConfig6/IfConfig6.h File Reference

Data Structures

struct _IFCONFIG6_INTERFACE_CB
struct _ARG_LIST
struct _IFCONFIG6_PRIVATE_DATA
struct _VAR_CHECK_ITEM

Defines

"memItemRight" valign="bottom">MACADDRMAXSIZE 32
#define PREFIXMAXLEN 16

Typedefs

typedef struct _IFCONFIG6_INTERFACE_CB IFCONFIG6_INTERFACE_CB
typedef struct _ARG_LIST ARG_LIST
typedef struct _IFCONFIG6_PRIVATE_DATA IFCONFIG6_PRIVATE_DATA
typedef struct _VAR_CHECK_ITEM VAR_CHECK_ITEM

Enumerations

enum { IfConfig6OpList = 1, IfConfig6OpSet = 2, IfConfig6OpClear = 3 }
enum VAR_CHECK_CODE { VarCheckReserved = -1, VarCheckOk = 0, VarCheckDuplicate, VarCheckConflict, VarCheckUnknown, VarCheckLackValue, VarCheckOutOfMem }
enum VAR_CHECK_FLAG_TYPE { FlagTypeSingle = 0, FlagTypeNeedVar, FlagTypeNeedSet, FlagTypeSkipUnknown }

Detailed Description

The interface function declaration of shell application IfConfig6.

Copyright (c) 2009 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by IfConfig6PrintMacAddr().

#define PREFIXMAXLEN 16

Referenced by IfConfig6PrintIpAddr().

Typedef Documentation

typedef struct _ARG_LIST ARG_LIST

typedef struct _IFCONFIG6_INTERFACE_CB IFCONFIG6_INTERFACE_CB

typedef struct _IFCONFIG6_PRIVATE_DATA IFCONFIG6_PRIVATE_DATA

typedef struct _VAR_CHECK_ITEM VAR_CHECK_ITEM

Enumeration Type Documentation

anonymous enum

Enumerator:

IfConfig6OpList
IfConfig6OpSet
IfConfig6OpClear

enum VAR_CHECK_CODE

Enumerator:

VarCheckReserved
VarCheckOk
VarCheckDuplicate
VarCheckConflict
VarCheckUnknown
VarCheckLackValue
VarCheckOutOfMem

enum VAR_CHECK_FLAG_TYPE

Enumerator:

FlagTypeSingle
FlagTypeNeedVar
FlagTypeNeedSet
FlagTypeSkipUnknown

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Application/IpsecConfig/Delete.c File Reference

Functions

EFI_STATUS DeletePolicyEntry (IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data, IN DELETE_POLICY_ENTRY_CONTEXT *Context)

EFI_STATUS FlushOrDeletePolicyEntry (IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN LIST_ENTRY *ParamPackage)

Detailed Description

The implementation of delete policy entry function in IpSecConfig application.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS DeletePolicyEntry ( IN EFI_IPSEC_CONFIG_SELECTOR *          Selector,
                               IN VOID *                               Data,
                               IN DELETE_POLICY_ENTRY_CONTEXT *        Context
                             )
```

Private function to delete entry information in database.

Parameters:

[in]	*Selector*	The pointer to EFI_IPSEC_CONFIG_SELECTOR structure.
[in]	*Data*	The pointer to Data.
[in]	*Context*	The pointer to DELETE_POLICY_ENTRY_CONTEXT.

Return values:

EFI_ABORTED	Abort the iteration.
EFI_SUCCESS	Continue the iteration.

References mIpSecConfig, and mMatchPolicyEntry.

Referenced by FlushOrDeletePolicyEntry().

```
EFI_STATUS FlushOrDeletePolicyEntry ( IN EFI_IPSEC_CONFIG_DATA_TYPE  DataType,
```

IN LIST_ENTRY * *ParamPackage*
)

Flush or delete entry information in the database according to datatype.

Parameters:

[in]	*DataType*	The value of EFI_IPSEC_CONFIG_DATA_TYPE.
[in]	*ParamPackage*	The pointer to the ParamPackage list.

Return values:

EFI_SUCCESS	Delete entry information successfully.
EFI_NOT_FOUND	Can't find the specified entry.
Others	Some mistaken case.

References DELETE_POLICY_ENTRY_CONTEXT::DataType, DeletePolicyEntry(), ForeachPolicyEntry(), DELETE_POLICY_ENTRY_CONTEXT::Indexer, mAppName, mConstructPolicyEntryIndexer, mHiiHandle, mIpSecConfig, and DELETE_POLICY_ENTRY_CONTEXT::Status.

Referenced by InitializeIpSecConfig().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Application/IpsecConfig/Delete.h File Reference

Data Structures

struct DELETE_POLICY_ENTRY_CONTEXT

Functions

EFI_STATUS FlushOrDeletePolicyEntry (IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN LIST_ENTRY *ParamPackage)

Detailed Description

The internal structure and function declaration of delete policy entry function in IpSecConfig application.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS FlushOrDeletePolicyEntry ( IN EFI_IPSEC_CONFIG_DATA_TYPE  DataType,
                                      IN LIST_ENTRY *                ParamPackage
                                    )
```

Flush or delete entry information in the database according to datatype.

Parameters:

[in]	*DataType*	The value of EFI_IPSEC_CONFIG_DATA_TYPE.
[in]	*ParamPackage*	The pointer to the ParamPackage list.

Return values:

EFI_SUCCESS	Delete entry information successfully.
EFI_NOT_FOUND	Can't find the specified entry.
Others	Some mistaken case.

References DELETE_POLICY_ENTRY_CONTEXT::DataType, DeletePolicyEntry(), ForeachPolicyEntry(), DELETE_POLICY_ENTRY_CONTEXT::Indexer, mAppName, mConstructPolicyEntryIndexer, mHiiHandle, mIpSecConfig, and DELETE_POLICY_ENTRY_CONTEXT::Status.

Referenced by InitializeIpSecConfig().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Application/IpsecConfig/Dump.c File Reference

Functions

UINTN GetVerFromAddrInfo (IN EFI_IP_ADDRESS_INFO *AddressInfo)
UINTN GetVerFromIpAddr (IN EFI_IP_ADDRESS *Address)
VOID DumpAsciiString (IN CHAR8 *Str, IN UINTN Length)
VOID DumpBuf (IN UINT8 *Data, IN UINTN Length)
VOID DumpAddressInfo (IN EFI_IP_ADDRESS_INFO *AddressInfo)
VOID DumpIpAddress (IN EFI_IP_ADDRESS *IpAddress)
VOID DumpSpdSelector (IN EFI_IPSEC_SPD_SELECTOR *Selector)
EFI_STATUS DumpSpdEntry (IN EFI_IPSEC_SPD_SELECTOR *Selector, IN EFI_IPSEC_SPD_DATA *Data, IN UINTN *EntryIndex)
EFI_STATUS DumpSadEntry (IN EFI_IPSEC_SA_ID *SaId, IN EFI_IPSEC_SA_DATA2 *Data, IN UINTN *EntryIndex)
EFI_STATUS DumpPadEntry (IN EFI_IPSEC_PAD_ID *PadId, IN EFI_IPSEC_PAD_DATA *Data, IN UINTN *EntryIndex)
EFI_STATUS ListPolicyEntry (IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN LIST_ENTRY *ParamPackage)

Variables

VISIT_POLICY_ENTRY mDumpPolicyEntry []

Detailed Description

The implementation of dump policy entry function in IpSecConfig application.

Copyright (c) 2009 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

VOID DumpAddressInfo (IN EFI_IP_ADDRESS_INFO * *AddressInfo*)
Private function called to print EFI_IP_ADDRESS_INFO content.

Parameters:

`[in]` *AddressInfo* The pointer to the EFI_IP_ADDRESS_INFO structure.

References GetVerFromAddrInfo().

Referenced by DumpPadEntry(), and DumpSpdSelector().

```
VOID DumpAsciiString ( IN CHAR8 * Str,
                       IN UINTN   Length
                     )
```

Private function called to print an ASCII string in unicode char format.

Parameters:

[in]	*Str*	The pointer to the ASCII string.
[in]	*Length*	The value of the ASCII string length.

Referenced by DumpPadEntry(), and DumpSadEntry().

```
VOID DumpBuf ( IN UINT8 * Data,
               IN UINTN   Length
             )
```

Private function called to print a buffer in Hex format.

Parameters:

[in]	*Data*	The pointer to the buffer.
[in]	*Length*	The size of the buffer.

Referenced by DumpSadEntry().

```
VOID DumpIpAddress ( IN EFI_IP_ADDRESS * IpAddress )
```

Private function called to print EFI_IP_ADDRESS content.

Parameters:

[in]	*IpAddress*	The pointer to the EFI_IP_ADDRESS structure.

References GetVerFromIpAddr().

Referenced by DumpSadEntry(), and DumpSpdEntry().

```
EFI_STATUS DumpPadEntry ( IN EFI_IPSEC_PAD_ID *      PadId,
                          IN EFI_IPSEC_PAD_DATA *    Data,
                          IN UINTN *                 EntryIndex
                        )
```

Print EFI_IPSEC_PAD_ID and EFI_IPSEC_PAD_DATA content.

Parameters:

[in]	*PadId*	The pointer to the EFI_IPSEC_PAD_ID structure.
[in]	*Data*	The pointer to the EFI_IPSEC_PAD_DATA structure.
[in]	*EntryIndex*	The pointer to the Index in the PAD Database.

Return values:

EFI_SUCCESS Dump PAD information successfully.

References DumpAddressInfo(), DumpAsciiString(), MapIntegerToString(), mMapAuthMethod, and mMapAuthProto.

```
EFI_STATUS DumpSadEntry ( IN EFI_IPSEC_SA_ID *        SaId,
                          IN EFI_IPSEC_SA_DATA2 *     Data,
                          IN UINTN *                  EntryIndex
                        )
```

Print EFI_IPSEC_SA_ID and EFI_IPSEC_SA_DATA2 content.

Parameters:

[in]	*SaId*	The pointer to the EFI_IPSEC_SA_ID structure.
[in]	*Data*	The pointer to the EFI_IPSEC_SA_DATA2 structure.
[in]	*EntryIndex*	The pointer to the Index in the SAD Database.

Return values:

EFI_SUCCESS Dump SAD information successfully.

References DumpAsciiString(), DumpBuf(), DumpIpAddress(), DumpSpdSelector(), MapIntegerToString(), mMapAuthAlgo, mMapEncAlgo, mMapIpSecMode, and mMapIpSecProtocol.

```
EFI_STATUS DumpSpdEntry ( IN EFI_IPSEC_SPD_SELECTOR *   Selector,
                          IN EFI_IPSEC_SPD_DATA *       Data,
                          IN UINTN *                    EntryIndex
                        )
```

Print EFI_IPSEC_SPD_SELECTOR and EFI_IPSEC_SPD_DATA content.

Parameters:

[in]	*Selector*	The pointer to the EFI_IPSEC_SPD_SELECTOR structure.
[in]	*Data*	The pointer to the EFI_IPSEC_SPD_DATA structure.
[in]	*EntryIndex*	The pointer to the Index in SPD Database.

Return values:

EFI_SUCCESS Dump SPD information successfully.

References DumpIpAddress(), DumpSpdSelector(), MapIntegerToString(), mMapAuthAlgo, mMapDfOption, mMapEncAlgo, mMapIpSecAction, mMapIpSecMode, and mMapIpSecProtocol.

```
VOID DumpSpdSelector ( IN EFI_IPSEC_SPD_SELECTOR *   Selector   )
```

Private function called to print EFI_IPSEC_SPD_SELECTOR content.

Parameters:

[in]	*Selector*	The pointer to the EFI_IPSEC_SPD_SELECTOR structure.

References DumpAddressInfo(), EFI_IP4_PROTO_ICMP, EFI_IP4_PROTO_TCP, EFI_IP4_PROTO_UDP, EFI_IPSEC_ANY_PORT, MapIntegerToString(), and mMapIpProtocol.

Referenced by DumpSadEntry(), and DumpSpdEntry().

UINTN GetVerFromAddrInfo (IN EFI_IP_ADDRESS_INFO * *AddressInfo*)

Private function called to get the version infomation from an EFI_IP_ADDRESS_INFO structure.

Parameters:

[in] *AddressInfo* The pointer to the EFI_IP_ADDRESS_INFO structure.

Returns:

the value of version.

Referenced by DumpAddressInfo().

UINTN GetVerFromIpAddr (IN EFI_IP_ADDRESS * *Address*)

Private function called to get the version information from a EFI_IP_ADDRESS structure.

Parameters:

[in] *Address* The pointer to the EFI_IP_ADDRESS structure.

Returns:

The value of the version.

Referenced by DumpIpAddress().

EFI_STATUS ListPolicyEntry (IN EFI_IPSEC_CONFIG_DATA_TYPE *DataType*, IN LIST_ENTRY * *ParamPackage*)

Print all entry information in the database according to datatype.

Parameters:

[in] *DataType* The value of EFI_IPSEC_CONFIG_DATA_TYPE.

[in] *ParamPackage* The pointer to the ParamPackage list.

Return values:

EFI_SUCCESS Dump all information successfully.

Others Some mistaken case.

References ForeachPolicyEntry(), and mDumpPolicyEntry.

Referenced by InitializeIpSecConfig().

Variable Documentation

VISIT_POLICY_ENTRY mDumpPolicyEntry[]

Initial value:

```
{
 (VISIT_POLICY_ENTRY) DumpSpdEntry,
```

```
  (VISIT_POLICY_ENTRY) DumpSadEntry,
  (VISIT_POLICY_ENTRY) DumpPadEntry
}
```

Referenced by ListPolicyEntry().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Application/IpsecConfig/Dump.h File Reference

Functions

EFI_STATUS ListPolicyEntry (IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN LIST_ENTRY *ParamPackage)

Detailed Description

The function declaration of dump policy entry function in IpSecConfig application.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS ListPolicyEntry ( IN EFI_IPSEC_CONFIG_DATA_TYPE  DataType,
                             IN LIST_ENTRY *                ParamPackage
                           )
```

Print all entry information in the database according to datatype.

Parameters:

[in]	*DataType*	The value of EFI_IPSEC_CONFIG_DATA_TYPE.
[in]	*ParamPackage*	The pointer to the ParamPackage list.

Return values:

EFI_SUCCESS	Dump all information successfully.
Others	Some mistaken case.

References ForeachPolicyEntry(), and mDumpPolicyEntry.

Referenced by InitializeIpSecConfig().

- Globals

NetworkPkg/Application/IpsecConfig/ForEach.c File Reference

Functions

EFI_STATUS ForeachPolicyEntry (IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN VISIT_POLICY_ENTRY Routine, IN VOID *Context)

Detailed Description

The implementation to go through each entry in IpSecConfig application.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS ForeachPolicyEntry ( IN EFI_IPSEC_CONFIG_DATA_TYPE  DataType,
                                IN VISIT_POLICY_ENTRY          Routine,
                                IN VOID *                      Context
                              )
```

Enumerate all entries in the database to execute specified operations according to datatype.

Parameters:

[in]	*DataType*	The value of EFI_IPSEC_CONFIG_DATA_TYPE.
[in]	*Routine*	The pointer to the function of a specified operation.
[in]	*Context*	The pointer to the context of a function.

Return values:

EFI_SUCCESS	Execute specified operation successfully.

References mIpSecConfig.

Referenced by AddOrInsertPolicyEntry(), EditPolicyEntry(), FlushOrDeletePolicyEntry(), and ListPolicyEntry().

- File List
- Globals

NetworkPkg/Application/IpsecConfig/ForEach.h File Reference

Typedefs

typedef EFI_STATUS(* VISIT_POLICY_ENTRY)(IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data, IN VOID *Context)

Functions

EFI_STATUS ForeachPolicyEntry (IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN VISIT_POLICY_ENTRY Routine, IN VOID *Context)

Detailed Description

The internal structure and function declaration of the implementation to go through each entry in IpSecConfig application.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Typedef Documentation

typedef EFI_STATUS(* VISIT_POLICY_ENTRY)(IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data, IN VOID *Context)

The prototype for the DumpSpdEntry()/DumpSadEntry()/DumpPadEntry(). Print EFI_IPSEC_CONFIG_SELECTOR and corresponding content.

Parameters:

[in]	*Selector*	The pointer to the EFI_IPSEC_CONFIG_SELECTOR union.
[in]	*Data*	The pointer to the corresponding data.
[in]	*Context*	The pointer to the Index in SPD/SAD/PAD Database.

Return values:

EFI_SUCCESS	Dump SPD/SAD/PAD information successfully.

Function Documentation

EFI_STATUS ForeachPolicyEntry (IN EFI_IPSEC_CONFIG_DATA_TYPE *DataType*,

```
IN VISIT_POLICY_ENTRY        Routine,
IN VOID *                    Context
)
```

Enumerate all entry in the database to execute a specified operation according to datatype.

Parameters:

[in]	*DataType*	The value of EFI_IPSEC_CONFIG_DATA_TYPE.
[in]	*Routine*	The pointer to function of a specified operation.
[in]	*Context*	The pointer to the context of a function.

Return values:

EFI_SUCCESS Execute specified operation successfully.

Enumerate all entries in the database to execute specified operations according to datatype.

Parameters:

[in]	*DataType*	The value of EFI_IPSEC_CONFIG_DATA_TYPE.
[in]	*Routine*	The pointer to the function of a specified operation.
[in]	*Context*	The pointer to the context of a function.

Return values:

EFI_SUCCESS Execute specified operation successfully.

References mIpSecConfig.

Referenced by AddOrInsertPolicyEntry(), EditPolicyEntry(), FlushOrDeletePolicyEntry(), and ListPolicyEntry().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Application/IpsecConfig/Helper.c File Reference

Functions

EFI_STATUS GetNumber (IN CHAR16 *FlagStr, IN UINT64 Maximum, IN OUT VOID *ValuePtr, IN UINTN ByteCount, IN STR2INT *Map, IN LIST_ENTRY *ParamPackage, IN UINT32 FormatMask)
EFI_STATUS EfiInetAddr2 (IN CHAR16 *Ptr, OUT EFI_IP_ADDRESS *Ip)
EFI_STATUS EfiInetAddrRange (IN CHAR16 *Ptr, OUT EFI_IP_ADDRESS_INFO *Addr)
EFI_STATUS EfiInetPortRange (IN CHAR16 *Ptr, OUT UINT16 *Port, OUT UINT16 *PortRange)
UINT64 StrToUInteger (IN CONST CHAR16 *Str, OUT EFI_STATUS *Status)
UINT32 MapStringToInteger (IN CONST CHAR16 *Str, IN STR2INT *Map)
CHAR16 * MapIntegerToString (IN UINT32 Integer, IN STR2INT *Map)

Detailed Description

The assistant function implementation for IpSecConfig application.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EfiInetAddr2 ( IN CHAR16 *              Ptr,
                          OUT EFI_IP_ADDRESS *     Ip
                        )
```

Helper function called to convert a string containing an Ipv4 or Ipv6 Internet Protocol address into a proper address for the EFI_IP_ADDRESS structure.

Parameters:

[in]	*Ptr*	The pointer to the string containing an Ipv4 or Ipv6 Internet Protocol address.
[out]	*Ip*	The pointer to the EFI_IP_ADDRESS structure to contain the result.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	Invalid parameter.

Referenced by ConstructSadIndexer(), CreateSadEntry(), and CreateSpdEntry().

```
EFI_STATUS EfiInetAddrRange ( IN CHAR16 *                Ptr,
                              OUT EFI_IP_ADDRESS_INFO *  Addr
                            )
```

Helper function called to calculate the prefix length associated with the string containing an Ipv4 or Ipv6 Internet Protocol address.

Parameters:

[in]	*Ptr*	The pointer to the string containing an Ipv4 or Ipv6 Internet Protocol address.
[out]	*Addr*	The pointer to the EFI_IP_ADDRESS_INFO structure to contain the result.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	Invalid parameter.
Others	Other mistake case.

Referenced by ConstructPadIndexer(), CreatePadEntry(), and CreateSpdSelector().

```
EFI_STATUS EfiInetPortRange ( IN CHAR16 *    Ptr,
                              OUT UINT16 *   Port,
                              OUT UINT16 *   PortRange
                            )
```

Helper function called to calculate the port range associated with the string.

Parameters:

[in]	*Ptr*	The pointer to the string containing a port and range.
[out]	*Port*	The pointer to the Port to contain the result.
[out]	*PortRange*	The pointer to the PortRange to contain the result.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	Invalid parameter.
Others	Other mistake case.

References StrToUInteger().

Referenced by CreateSpdSelector().

```
EFI_STATUS GetNumber ( IN CHAR16 *        FlagStr,
                       IN UINT64          Maximum,
                       IN OUT VOID *      ValuePtr,
                       IN UINTN           ByteCount,
                       IN STR2INT *       Map,
                       IN LIST_ENTRY *    ParamPackage,
                       IN UINT32          FormatMask
                     )
```

Helper function called to change an input parameter in the string format to a number.

Parameters:

[in]	*FlagStr*	The pointer to the flag string.
[in]	*Maximum*	Greatest value number.
[in,out]	*ValuePtr*	The pointer to the input parameter in string format.
[in]	*ByteCount*	The valid byte count
[in]	*Map*	The pointer to the STR2INT table.
[in]	*ParamPackage*	The pointer to the ParamPackage list.
[in]	*FormatMask*	The bit mask. BIT 0 set indicates the value of a flag might be a number. BIT 1 set indicates the value of a flag might be a string that needs to be looked up.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_FOUND	The input parameter can't be found.
EFI_INVALID_PARAMETER	The input parameter is an invalid input.

References FORMAT_NUMBER, FORMAT_STRING, mAppName, MapStringToInteger(), mHiiHandle, and StrToUInteger().

Referenced by ConstructSadIndexer(), CreatePadEntry(), CreateSadEntry(), CreateSpdEntry(), and CreateSpdSelector().

```
CHAR16* MapIntegerToString ( IN UINT32        Integer,
                             IN STR2INT *     Map
                           )
```

Helper function called to transfer an unsigned integer to a string according to the map table.

Parameters:

[in]	*Integer*	The pointer to the string.
[in]	*Map*	The pointer to the map table.

Returns:

The converted Str. If not found, then return NULL.

References STR2INT::Integer, and STR2INT::String.

Referenced by DumpPadEntry(), DumpSadEntry(), DumpSpdEntry(), and DumpSpdSelector().

```
UINT32 MapStringToInteger ( IN CONST CHAR16 *    Str,
                            IN STR2INT *         Map
                          )
```

Helper function called to transfer a string to an unsigned integer according to the map table.

Parameters:

[in]	*Str*	The pointer to the string.
[in]	*Map*	The pointer to the map table.

Returns:

The integer value of converted Str. If not found, then return -1.

References STR2INT::Integer, and STR2INT::String.

Referenced by GetNumber(), and InitializeIpSecConfig().

```
UINT64 StrToUInteger ( IN CONST CHAR16 *    Str,
                       OUT EFI_STATUS *     Status
                     )
```

Helper function called to transfer a string to an unsigned integer.

Parameters:

[in]	*Str*	The pointer to the string.
[out]	*Status*	The operation status.

Returns:

The integer value of converted Str.

Referenced by ConstructPadIndexer(), ConstructSadIndexer(), ConstructSpdIndexer(), EfiInetPortRange(), and GetNumber().

NetworkPkg/Application/IpsecConfig/Helper.h File Reference

Defines

"memItemRight" valign="bottom">FORMAT_NUMBER 0x1

#define FORMAT_STRING 0x2

Functions

EFI_STATUS GetNumber (IN CHAR16 *FlagStr, IN UINT64 Maximum, IN OUT VOID *ValuePtr, IN UINTN ByteCount, IN STR2INT *Map, IN LIST_ENTRY *ParamPackage, IN UINT32 FormatMask)

EFI_STATUS EfiInetAddr2 (IN CHAR16 *Ptr, OUT EFI_IP_ADDRESS *Ip)

EFI_STATUS EfiInetAddrRange (IN CHAR16 *Ptr, OUT EFI_IP_ADDRESS_INFO *Addr)

EFI_STATUS EfiInetPortRange (IN CHAR16 *Ptr, OUT UINT16 *Port, OUT UINT16 *PortRange)

UINT64 StrToUInteger (IN CONST CHAR16 *Str, OUT EFI_STATUS *Status)

UINT32 MapStringToInteger (IN CONST CHAR16 *Str, IN STR2INT *Map)

CHAR16 * MapIntegerToString (IN UINT32 Integer, IN STR2INT *Map)

Detailed Description

The assistant function declaration for IpSecConfig application.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by ConstructSadIndexer(), CreateSadEntry(), CreateSpdEntry(), CreateSpdSelector(), and GetNumber().

#define FORMAT_STRING 0x2

Referenced by ConstructSadIndexer(), CreatePadEntry(), CreateSadEntry(), CreateSpdEntry(), CreateSpdSelector(), and GetNumber().

Function Documentation

```
EFI_STATUS EfiInetAddr2 ( IN CHAR16 *              Ptr,
                          OUT EFI_IP_ADDRESS *     Ip
                        )
```

Helper function called to convert a string containing an (Ipv4) Internet Protocol dotted address into a proper address for the EFI_IP_ADDRESS structure.

Parameters:

[in]	*Ptr*	The pointer to the string containing an (Ipv4) Internet Protocol dotted address.
[out]	*Ip*	The pointer to the Ip address structure to contain the result.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	Invalid parameter.

Helper function called to convert a string containing an Ipv4 or Ipv6 Internet Protocol address into a proper address for the EFI_IP_ADDRESS structure.

Parameters:

[in]	*Ptr*	The pointer to the string containing an Ipv4 or Ipv6 Internet Protocol address.
[out]	*Ip*	The pointer to the EFI_IP_ADDRESS structure to contain the result.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	Invalid parameter.

Referenced by ConstructSadIndexer(), CreateSadEntry(), and CreateSpdEntry().

```
EFI_STATUS EfiInetAddrRange ( IN CHAR16 *                Ptr,
                              OUT EFI_IP_ADDRESS_INFO *  Addr
                            )
```

Helper function called to calculate the prefix length associated with the string containing an Ipv4 or Ipv6 Internet Protocol address.

Parameters:

[in]	*Ptr*	The pointer to the string containing an Ipv4 or Ipv6 Internet Protocol address.
[out]	*Addr*	The pointer to the EFI_IP_ADDRESS_INFO structure to contain the result.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	Invalid parameter.
Others	Other mistake case.

Referenced by ConstructPadIndexer(), CreatePadEntry(), and CreateSpdSelector().

```
EFI_STATUS EfiInetPortRange ( IN CHAR16 *     Ptr,
                              OUT UINT16 *    Port,
                              OUT UINT16 *    PortRange
                            )
```

Helper function called to calculate the port range associated with the string.

Parameters:

[in]	*Ptr*	The pointer to the string containing a port and range.
[out]	*Port*	The pointer to the Port to contain the result.
[out]	*PortRange*	The pointer to the PortRange to contain the result.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	Invalid parameter.
Others	Other mistake case.

References StrToUInteger().

Referenced by CreateSpdSelector().

```
EFI_STATUS GetNumber ( IN CHAR16 *        FlagStr,
                       IN UINT64          Maximum,
                       IN OUT VOID *      ValuePtr,
                       IN UINTN           ByteCount,
                       IN STR2INT *       Map,
                       IN LIST_ENTRY *    ParamPackage,
                       IN UINT32          FormatMask
                     )
```

Helper function called to change input parameter in string format to number.

Parameters:

[in]	*FlagStr*	The pointer to the flag string.
[in]	*Maximum*	most value number.
[in,out]	*ValuePtr*	The pointer to the input parameter in string format.
[in]	*ByteCount*	The valid byte count
[in]	*Map*	The pointer to the STR2INT table.
[in]	*ParamPackage*	The pointer to the ParamPackage list.
[in]	*FormatMask*	The bit mask. BIT 0 set indicates the value of flag might be number. BIT 1 set indicates the value of flag might be a string that needs to be looked up.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_FOUND	The input parameter can't be found.
EFI_INVALID_PARAMETER	The input parameter is an invalid input.

Helper function called to change an input parameter in the string format to a number.

Parameters:

[in]	*FlagStr*	The pointer to the flag string.

[in]	*Maximum*	Greatest value number.
[in,out]	*ValuePtr*	The pointer to the input parameter in string format.
[in]	*ByteCount*	The valid byte count
[in]	*Map*	The pointer to the STR2INT table.
[in]	*ParamPackage*	The pointer to the ParamPackage list.
[in]	*FormatMask*	The bit mask. BIT 0 set indicates the value of a flag might be a number. BIT 1 set indicates the value of a flag might be a string that needs to be looked up.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_FOUND	The input parameter can't be found.
EFI_INVALID_PARAMETER	The input parameter is an invalid input.

References FORMAT_NUMBER, FORMAT_STRING, mAppName, MapStringToInteger(), mHiiHandle, and StrToUInteger().

Referenced by ConstructSadIndexer(), CreatePadEntry(), CreateSadEntry(), CreateSpdEntry(), and CreateSpdSelector().

```
CHAR16* MapIntegerToString ( IN UINT32      Integer,
                             IN STR2INT *   Map
                           )
```

Helper function called to transfer an unsigned integer to a string according to the map table.

Parameters:

[in]	*Integer*	The pointer to the string.
[in]	*Map*	The pointer to the map table.

Returns:

The converted str. If not found, then return NULL.

Helper function called to transfer an unsigned integer to a string according to the map table.

Parameters:

[in]	*Integer*	The pointer to the string.
[in]	*Map*	The pointer to the map table.

Returns:

The converted Str. If not found, then return NULL.

References STR2INT::Integer, and STR2INT::String.

Referenced by DumpPadEntry(), DumpSadEntry(), DumpSpdEntry(), and DumpSpdSelector().

```
UINT32 MapStringToInteger ( IN CONST CHAR16 *   Str,
                            IN STR2INT *        Map
                          )
```

Helper function called to transfer a string to an unsigned integer according to the map table.

Parameters:

[in] *Str* The pointer to the string.
[in] *Map* The pointer to the map table.

Returns:

The integer value of converted str. If not found, then return -1.

Helper function called to transfer a string to an unsigned integer according to the map table.

Parameters:

[in] *Str* The pointer to the string.
[in] *Map* The pointer to the map table.

Returns:

The integer value of converted Str. If not found, then return -1.

References STR2INT::Integer, and STR2INT::String.

Referenced by GetNumber(), and InitializeIpSecConfig().

```
UINT64 StrToUInteger ( IN CONST CHAR16 *    Str,
                       OUT EFI_STATUS *     Status
                     )
```

Helper function called to transfer a string to an unsigned integer.

Parameters:

[in] *Str* The pointer to the string.
[out] *Status* The operation status.

Returns:

The integer value of a converted str.

Helper function called to transfer a string to an unsigned integer.

Parameters:

[in] *Str* The pointer to the string.
[out] *Status* The operation status.

Returns:

The integer value of converted Str.

Referenced by ConstructPadIndexer(), ConstructSadIndexer(), ConstructSpdIndexer(), EfiInetPortRange(), and GetNumber().

NetworkPkg/Application/IpsecConfig/Indexer.c File Reference

Functions

EFI_STATUS ConstructSpdIndexer (IN OUT SPD_ENTRY_INDEXER *Indexer, IN LIST_ENTRY *ParamPackage)
EFI_STATUS ConstructSadIndexer (IN OUT SAD_ENTRY_INDEXER *Indexer, IN LIST_ENTRY *ParamPackage)
EFI_STATUS ConstructPadIndexer (IN OUT PAD_ENTRY_INDEXER *Indexer, IN LIST_ENTRY *ParamPackage)

Variables

CONSTRUCT_POLICY_ENTRY_INDEXER mConstructPolicyEntryIndexer []

Detailed Description

The implementation of construct ENTRY_INDEXER in IpSecConfig application.

Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS ConstructPadIndexer (IN OUT PAD_ENTRY_INDEXER * *Indexer*,
IN LIST_ENTRY * *ParamPackage*
)

Fill in PAD_ENTRY_INDEXER through ParamPackage list.

Parameters:

[in,out]	*Indexer*	The pointer to the PAD_ENTRY_INDEXER structure.
[in]	*ParamPackage*	The pointer to the ParamPackage list.

Return values:

EFI_SUCCESS	Filled in PAD_ENTRY_INDEXER successfully.
EFI_INVALID_PARAMETER	The mistaken user input in ParamPackage list.

References EfiInetAddrRange(), mAppName, mHiiHandle, and StrToUInteger().

EFI_STATUS ConstructSadIndexer (IN OUT SAD_ENTRY_INDEXER * *Indexer*,
IN LIST_ENTRY * *ParamPackage*
)

Fill in SAD_ENTRY_INDEXER through ParamPackage list.

Parameters:

[in,out]	*Indexer*	The pointer to the SAD_ENTRY_INDEXER structure.
[in]	*ParamPackage*	The pointer to the ParamPackage list.

Return values:

EFI_SUCCESS	Filled in SPD_ENTRY_INDEXER successfully.
EFI_INVALID_PARAMETER	The mistaken user input in ParamPackage list.

References EfiInetAddr2(), FORMAT_NUMBER, FORMAT_STRING, GetNumber(), mAppName, mHiiHandle, mMapIpSecProtocol, and StrToUInteger().

EFI_STATUS ConstructSpdIndexer (IN OUT SPD_ENTRY_INDEXER * *Indexer*,
IN LIST_ENTRY * *ParamPackage*
)

Fill in SPD_ENTRY_INDEXER through ParamPackage list.

Parameters:

[in,out]	*Indexer*	The pointer to the SPD_ENTRY_INDEXER structure.
[in]	*ParamPackage*	The pointer to the ParamPackage list.

Return values:

EFI_SUCCESS	Filled in SPD_ENTRY_INDEXER successfully.

References StrToUInteger().

Variable Documentation

CONSTRUCT_POLICY_ENTRY_INDEXER mConstructPolicyEntryIndexer[]

Initial value:

```
{
  (CONSTRUCT_POLICY_ENTRY_INDEXER) ConstructSpdIndexer,
  (CONSTRUCT_POLICY_ENTRY_INDEXER) ConstructSadIndexer,
  (CONSTRUCT_POLICY_ENTRY_INDEXER) ConstructPadIndexer
}
```

Referenced by AddOrInsertPolicyEntry(), EditPolicyEntry(), and FlushOrDeletePolicyEntry().

NetworkPkg/Application/IpsecConfig/Indexer.h File Reference

Data Structures

struct	SPD_ENTRY_INDEXER
struct	SAD_ENTRY_INDEXER
struct	PAD_ENTRY_INDEXER
union	POLICY_ENTRY_INDEXER

Typedefs

typedef EFI_STATUS(* CONSTRUCT_POLICY_ENTRY_INDEXER)(IN POLICY_ENTRY_INDEXER *Indexer, IN LIST_ENTRY *ParamPackage)

Variables

CONSTRUCT_POLICY_ENTRY_INDEXER mConstructPolicyEntryIndexer []

Detailed Description

The internal structure and function declaration to construct ENTRY_INDEXER in IpSecConfig application.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Typedef Documentation

typedef EFI_STATUS(* CONSTRUCT_POLICY_ENTRY_INDEXER)(IN POLICY_ENTRY_INDEXER *Indexer, IN LIST_ENTRY *ParamPackage)

The prototype for the ConstructSpdIndexer()/ConstructSadIndexer()/ConstructPadIndexer(). Fill in SPD_ENTRY_INDEXER/SAD_ENTRY_INDEXER/PAD_ENTRY_INDEXER through ParamPackage list.

Parameters:

`[in,out]`	*Indexer*	The pointer to the POLICY_ENTRY_INDEXER union.
`[in]`	*ParamPackage*	The pointer to the ParamPackage list.

Return values:

EFI_SUCCESS	Filled in POLICY_ENTRY_INDEXER successfully.

Variable Documentation

CONSTRUCT_POLICY_ENTRY_INDEXER mConstructPolicyEntryIndexer[]

Referenced by AddOrInsertPolicyEntry(), EditPolicyEntry(), and FlushOrDeletePolicyEntry().

NetworkPkg/Application/IpsecConfig/IpSecConfig.c File Reference

Functions

BOOLEAN GrowBuffer (IN OUT EFI_STATUS *Status, IN OUT VOID **Buffer, IN UINTN BufferSize)

EFI_STATUS LocateHandle (IN EFI_LOCATE_SEARCH_TYPE SearchType, IN EFI_GUID *Protocol, IN VOID *SearchKey, IN OUT UINTN *NoHandles, OUT EFI_HANDLE **Buffer)

EFI_STATUS LocateProtocol (IN EFI_GUID *ProtocolGuid, OUT VOID **Interface)

EFI_STATUS IpSecConfigRetriveCheckListByName (IN VAR_CHECK_ITEM *CheckList, IN LIST_ENTRY *ParamPackage)

EFI_STATUS EFIAPI InitializeIpSecConfig (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)

Variables

SHELL_PARAM_ITEM mIpSecConfigParamList []

STR2INT mMapPolicy []

STR2INT mMapIpProtocol []

STR2INT mMapIpSecAction []

STR2INT mMapIpSecMode []

STR2INT mMapDfOption []

STR2INT mMapIpSecProtocol []

STR2INT mMapAuthAlgo []

STR2INT mMapEncAlgo []

STR2INT mMapAuthProto []

STR2INT mMapAuthMethod []

EFI_IPSEC2_PROTOCOL * mIpSec

EFI_IPSEC_CONFIG_PROTOCOL * mIpSecConfig

EFI_HII_HANDLE mHiiHandle

CHAR16 mAppName [] = L"IpSecConfig"

VAR_CHECK_ITEM mIpSecConfigVarCheckList []

Detailed Description

The main process for IpSecConfig application.

Copyright (c) 2009 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
BOOLEAN GrowBuffer ( IN OUT EFI_STATUS *   Status,
                     IN OUT VOID **        Buffer,
                     IN UINTN              BufferSize
                   )
```

The function to allocate the proper sized buffer for various EFI interfaces.

Parameters:

[in,out]	*Status*	Current status.
[in,out]	*Buffer*	Current allocated buffer, or NULL.
[in]	*BufferSize*	Current buffer size needed

Return values:

TRUE	If the buffer was reallocated and the caller should try the API again.
FALSE	If the buffer was not reallocated successfully.

Referenced by LocateHandle().

```
EFI_STATUS EFIAPI InitializeIpSecConfig ( IN EFI_HANDLE          ImageHandle,
                                          IN EFI_SYSTEM_TABLE *  SystemTable
                                        )
```

This is the declaration of an EFI image entry point. This entry point is the same for UEFI Applications, UEFI OS Loaders, and UEFI Drivers, including both device drivers and bus drivers.

The entry point for IpSecConfig application that parse the command line input and call an IpSecConfig process.

Parameters:

[in]	*ImageHandle*	The image handle of this application.
[in]	*SystemTable*	The pointer to the EFI System Table.

Return values:

EFI_SUCCESS	The operation completed successfully.

References AddOrInsertPolicyEntry(), EditPolicyEntry(), FlushOrDeletePolicyEntry(), IPSEC_STATUS_ENABLED, IPSECCONFIG_STATUS_NAME, IpSecConfigRetriveCheckListByName(), ListPolicyEntry(), LocateProtocol(), mAppName, MapStringToInteger(), mHiiHandle, mIpSec, mIpSecConfig, and mIpSecConfigParamList.

```
EFI_STATUS IpSecConfigRetriveCheckListByName ( IN VAR_CHECK_ITEM *   CheckList,
                                               IN LIST_ENTRY *       ParamPackage
                                             )
```

Helper function called to check the conflicted flags.

Parameters:

[in]	*CheckList*	The pointer to the VAR_CHECK_ITEM table.
[in]	*ParamPackage*	The pointer to the ParamPackage list.

Return values:

EFI_SUCCESS	No conflicted flags.
EFI_INVALID_PARAMETER	The input parameter is erroroneous or there are some conflicted flags.

References VAR_CHECK_ITEM::Attribute1, VAR_CHECK_ITEM::Attribute2, VAR_CHECK_ITEM::Attribute3, VAR_CHECK_ITEM::Attribute4, and VAR_CHECK_ITEM::VarName.

Referenced by InitializeIpSecConfig().

```
EFI_STATUS LocateHandle ( IN EFI_LOCATE_SEARCH_TYPE  SearchType,
                          IN EFI_GUID *              Protocol,
                          IN VOID *                  SearchKey,
                          IN OUT UINTN *             NoHandles,
                          OUT EFI_HANDLE **          Buffer
                        )
```

Function returns an array of handles that support the requested protocol in a buffer allocated from a pool.

Parameters:

`[in]`	*SearchType*	Specifies which handle(s) are to be returned.
`[in]`	*Protocol*	Provides the protocol to search by. This parameter is only valid for SearchType ByProtocol.
`[in]`	*SearchKey*	Supplies the search key depending on the SearchType.
`[in,out]`	*NoHandles*	The number of handles returned in Buffer.
`[out]`	*Buffer*	A pointer to the buffer to return the requested array of handles that support Protocol.

Return values:

EFI_SUCCESS	The resulting array of handles was returned.
Others	Other mistake case.

References GrowBuffer().

Referenced by LocateProtocol().

```
EFI_STATUS LocateProtocol ( IN EFI_GUID *  ProtocolGuid,
                            OUT VOID **    Interface
                          )
```

Find the first instance of this protocol in the system and return its interface.

Parameters:

`[in]`	*ProtocolGuid*	The guid of the protocol.
`[out]`	*Interface*	The pointer to the first instance of the protocol.

Return values:

EFI_SUCCESS	A protocol instance matching ProtocolGuid was found.
Others	A protocol instance matching ProtocolGuid was not found.

References LocateHandle().

Referenced by InitializeIpSecConfig().

Variable Documentation

CHAR16 mAppName[] = L"IpSecConfig"

Referenced by AddOrInsertPolicyEntry(), CombineSadEntry(), CombineSpdEntry(), ConstructPadIndexer(), ConstructSadIndexer(), CreatePadEntry(), CreateSadEntry(), CreateSpdEntry(), CreateSpdSelector(), EditPolicyEntry(), FlushOrDeletePolicyEntry(), GetNumber(), and InitializeIpSecConfig().

EFI_HII_HANDLE mHiiHandle

EFI_IPSEC2_PROTOCOL* mIpSec

Referenced by InitializeIpSecConfig(), and Ip6IpSecProcessPacket().

EFI_IPSEC_CONFIG_PROTOCOL* mIpSecConfig

Referenced by AddOrInsertPolicyEntry(), DeletePolicyEntry(), EditOperatePolicyEntry(), FlushOrDeletePolicyEntry(), ForeachPolicyEntry(), InitializeIpSecConfig(), and InsertPolicyEntry().

SHELL_PARAM_ITEM mIpSecConfigParamList[]

Referenced by InitializeIpSecConfig().

VAR_CHECK_ITEM mIpSecConfigVarCheckList[]

STR2INT mMapAuthAlgo[]

Initial value:

```
{
  { L"NONE",          IPSEC_AALG_NONE },
  { L"MD5HMAC",       IPSEC_AALG_MD5HMAC },
  { L"SHA1HMAC",      IPSEC_AALG_SHA1HMAC },
  { L"SHA2-256HMAC", IPSEC_AALG_SHA2_256HMAC },
  { L"SHA2-384HMAC", IPSEC_AALG_SHA2_384HMAC },
  { L"SHA2-512HMAC", IPSEC_AALG_SHA2_512HMAC },
  { L"AES-XCBC-MAC", IPSEC_AALG_AES_XCBC_MAC },
  { L"NULL",          IPSEC_AALG_NULL },
  { NULL,             0 },
}
```

Referenced by CreateSadEntry(), CreateSpdEntry(), DumpSadEntry(), and DumpSpdEntry().

STR2INT mMapAuthMethod[]

Initial value:

```
{
  { L"PreSharedSecret", EfiIPsecAuthMethodPreSharedSecret },
  { L"Certificates",    EfiIPsecAuthMethodCertificates },
```

```
  { NULL,                0 },
}
```

Referenced by CreatePadEntry(), and DumpPadEntry().

STR2INT mMapAuthProto[]

Initial value:

```
 {
  { L"IKEv1",         EfiIPsecAuthProtocolIKEv1 },
  { L"IKEv2",         EfiIPsecAuthProtocolIKEv2 },
  { NULL,             0 },
}
```

Referenced by CreatePadEntry(), and DumpPadEntry().

STR2INT mMapDfOption[]

Initial value:

```
 {
  { L"clear",      EfiIPsecTunnelClearDf },
  { L"set",        EfiIPsecTunnelSetDf },
  { L"copy",       EfiIPsecTunnelCopyDf },
  { NULL,          0 },
}
```

Referenced by CreateSpdEntry(), and DumpSpdEntry().

STR2INT mMapEncAlgo[]

Initial value:

```
 {
  { L"NONE",          IPSEC_EALG_NONE },
  { L"DESCBC",        IPSEC_EALG_DESCBC },
  { L"3DESCBC",       IPSEC_EALG_3DESCBC },
  { L"CASTCBC",       IPSEC_EALG_CASTCBC },
  { L"BLOWFISHCBC",   IPSEC_EALG_BLOWFISHCBC },
  { L"NULL",          IPSEC_EALG_NULL },
  { L"AESCBC",        IPSEC_EALG_AESCBC },
  { L"AESCTR",        IPSEC_EALG_AESCTR },
  { L"AES-CCM-ICV8",  IPSEC_EALG_AES_CCM_ICV8 },
  { L"AES-CCM-ICV12",IPSEC_EALG_AES_CCM_ICV12 },
  { L"AES-CCM-ICV16",IPSEC_EALG_AES_CCM_ICV16 },
  { L"AES-GCM-ICV8",  IPSEC_EALG_AES_GCM_ICV8 },
  { L"AES-GCM-ICV12",IPSEC_EALG_AES_GCM_ICV12 },
  { L"AES-GCM-ICV16",IPSEC_EALG_AES_GCM_ICV16 },
  { NULL,             0 },
}
```

Referenced by CreateSadEntry(), CreateSpdEntry(), DumpSadEntry(), and DumpSpdEntry().

STR2INT mMapIpProtocol[]

Initial value:

```
 {
  { L"TCP",        EFI_IP4_PROTO_TCP },
  { L"UDP",        EFI_IP4_PROTO_UDP },
  { L"ICMP",       EFI_IP4_PROTO_ICMP },
  { NULL,          0 },
}
```

Referenced by CreateSpdSelector(), and DumpSpdSelector().

STR2INT mMapIpSecAction[]

Initial value:

```
 {
  { L"Bypass",     EfiIPsecActionBypass },
  { L"Discard",    EfiIPsecActionDiscard },
  { L"Protect",    EfiIPsecActionProtect },
  { NULL,          0 },
}
```

Referenced by CreateSpdEntry(), and DumpSpdEntry().

STR2INT mMapIpSecMode[]

Initial value:

```
 {
  { L"Transport", EfiIPsecTransport },
  { L"Tunnel",    EfiIPsecTunnel },
  { NULL,         0 },
}
```

Referenced by CreateSadEntry(), CreateSpdEntry(), DumpSadEntry(), and DumpSpdEntry().

STR2INT mMapIpSecProtocol[]

Initial value:

```
 {
  { L"AH",        EfiIPsecAH },
  { L"ESP",       EfiIPsecESP },
  { NULL,         0 },
}
```

Referenced by ConstructSadIndexer(), CreateSadEntry(), CreateSpdEntry(), DumpSadEntry(), and DumpSpdEntry().

STR2INT mMapPolicy[]

Initial value:

```
 {
  { L"SPD",       IPsecConfigDataTypeSpd },
  { L"SAD",       IPsecConfigDataTypeSad },
  { L"PAD",       IPsecConfigDataTypePad },
  { NULL,         0 },
}
```

NetworkPkg/Application/IpsecConfig/IpSecConfig.h File Reference

Data Structures

struct VAR_CHECK_ITEM
struct SHELL_PARAM_PACKAGE
struct STR2INT

Defines

"memItemRight" valign="bottom">ARRAY_SIZE(x) (sizeof(x) / sizeof(x[0]))
#define IPSECCONFIG_STATUS_NAME
#define BIT(x) (UINT32) (1 << (x))
#define IPSEC_STATUS_DISABLED 0x0
#define IPSEC_STATUS_ENABLED 0x1
#define EFI_IP4_PROTO_ICMP 0x1
#define EFI_IP4_PROTO_TCP 0x6
#define EFI_IP4_PROTO_UDP 0x11
#define EFI_IPSEC_ANY_PROTOCOL 0x
#define EFI_IPSEC_ANY_PORT 0
#define IPSEC_AALG_NONE 0x00
#define IPSEC_AALG_MD5HMAC 0x01
#define IPSEC_AALG_SHA1HMAC 0x02
#define IPSEC_AALG_SHA2_256HMAC
#define IPSEC_AALG_SHA2_384HMAC
#define IPSEC_AALG_SHA2_512HMAC
#define IPSEC_AALG_AES_XCBC_MAC
#define IPSEC_AALG_NULL 0xFB
#define IPSEC_EALG_NONE 0x00
#define IPSEC_EALG_DESCBC 0x02
#define IPSEC_EALG_3DESCBC 0x03
#define IPSEC_EALG_CASTCBC 0x06
#define IPSEC_EALG_BLOWFISHCBC 0
#define IPSEC_EALG_NULL 0x0B
#define IPSEC_EALG_AESCBC 0x0C
#define IPSEC_EALG_AESCTR 0x0D
#define IPSEC_EALG_AES_CCM_ICV8 0
#define IPSEC_EALG_AES_CCM_ICV12
#define IPSEC_EALG_AES_CCM_ICV16
#define IPSEC_EALG_AES_GCM_ICV8 0
#define IPSEC_EALG_AES_GCM_ICV12
#define IPSEC_EALG_AES_GCM_ICV16

Variables

EFI_IPSEC_CONFIG_PROTOCOL * mIpSecConfig
EFI_HII_HANDLE mHiiHandle
CHAR16 mAppName []
STR2INT mMapPolicy []

STR2INT mMapIpProtocol []
STR2INT mMapIpSecAction []
STR2INT mMapIpSecMode []
STR2INT mMapDfOption []
STR2INT mMapIpSecProtocol []
STR2INT mMapAuthAlgo []
STR2INT mMapEncAlgo []
STR2INT mMapAuthProto []
STR2INT mMapAuthMethod []

Detailed Description

The internal structure and function declaration in IpSecConfig application.

Copyright (c) 2009 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"paramtype">x) (sizeof(x) / sizeof(x[0]))

#define BIT (x) (UINT32) (1 << (x))

#define EFI_IP4_PROTO_ICMP 0x1

Referenced by CombineSadEntry(), CombineSpdEntry(), and DumpSpdSelector().

#define EFI_IP4_PROTO_TCP 0x6

Referenced by CombineSadEntry(), CombineSpdEntry(), and DumpSpdSelector().

#define EFI_IP4_PROTO_UDP 0x11

Referenced by CombineSadEntry(), CombineSpdEntry(), and DumpSpdSelector().

#define EFI_IPSEC_ANY_PORT 0

Referenced by CreateSpdSelector(), DumpSpdSelector(), Ikev2MatchSpdEntry(), and Ikev2NegotiateChildSa().

#define EFI_IPSEC_ANY_PROTOCOL 0xFFFF

Referenced by CreateSpdSelector(), Ikev2MatchSpdEntry(), Ikev2NegotiateChildSa(), and IpSecMatchNextLayerProtocol().

#define IPSEC_AALG_AES_XCBC_MAC 0x09

#define IPSEC_AALG_MD5HMAC 0x01

#define IPSEC_AALG_NONE 0x00

IPsec Authentication Algorithm Definition The number value definition is aligned to IANA assignment

Referenced by CreateSadEntry().

#define IPSEC_AALG_NULL 0xFB

#define IPSEC_AALG_SHA1HMAC 0x02

#define IPSEC_AALG_SHA2_256HMAC 0x05

#define IPSEC_AALG_SHA2_384HMAC 0x06

#define IPSEC_AALG_SHA2_512HMAC 0x07

#define IPSEC_EALG_3DESCBC 0x03

#define IPSEC_EALG_AES_CCM_ICV12 0x0F

#define IPSEC_EALG_AES_CCM_ICV16 0x10

#define IPSEC_EALG_AES_CCM_ICV8 0x0E

#define IPSEC_EALG_AES_GCM_ICV12 0x13

#define IPSEC_EALG_AES_GCM_ICV16 0x14

#define IPSEC_EALG_AES_GCM_ICV8 0x12

#define IPSEC_EALG_AESCBC 0x0C

#define IPSEC_EALG_AESCTR 0x0D

#define IPSEC_EALG_BLOWFISHCBC 0x07

#define IPSEC_EALG_CASTCBC 0x06

#define IPSEC_EALG_DESCBC 0x02

#define IPSEC_EALG_NONE 0x00
IPsec Encryption Algorithm Definition The number value definition is aligned to IANA assignment

Referenced by CombineSadEntry(), and CreateSadEntry().

#define IPSEC_EALG_NULL 0x0B

#define IPSEC_STATUS_DISABLED 0x0

Referenced by IkeDeleteAllSas(), Ikev2InfoParser(), Ikev2OnPacketSent(), and Ikev2ResendNotify().

#define IPSEC_STATUS_ENABLED 0x1

Referenced by InitializeIpSecConfig(), and IpSecConfigRestore().

#define IPSECCONFIG_STATUS_NAME L"IpSecStatus"

Referenced by IkeDeleteAllSas(), Ikev2InfoParser(), Ikev2OnPacketSent(), Ikev2ResendNotify(), InitializeIpSecConfig(), and IpSecConfigRestore().

Variable Documentation

CHAR16 mAppName[]

Referenced by AddOrInsertPolicyEntry(), CombineSadEntry(), CombineSpdEntry(), ConstructPadIndexer(), ConstructSadIndexer(), CreatePadEntry(), CreateSadEntry(), CreateSpdEntry(), CreateSpdSelector(), EditPolicyEntry(), FlushOrDeletePolicyEntry(), GetNumber(), and InitializeIpSecConfig().

EFI_HII_HANDLE mHiiHandle

Referenced by AddOrInsertPolicyEntry(), AddVlan(), CombineSadEntry(), CombineSpdEntry(), ConstructPadIndexer(), ConstructSadIndexer(), CreatePadEntry(), CreateSadEntry(), CreateSpdEntry(), CreateSpdSelector(), DeleteVlan(), EditPolicyEntry(), FlushOrDeletePolicyEntry(), GetNumber(), IfConfig6ClearInterfaceInfo(), IfConfig6GetInterfaceInfo(), IfConfig6Initialize(), IfConfig6PrintIpAddr(), IfConfig6PrintMacAddr(), IfConfig6SetInterfaceInfo(), IfConfig6ShowInterfaceInfo(), InitializeIpSecConfig(), InitializePing6(), LocateNicHandleBuffer(), NicNameToHandle(), Ping6(), Ping6CreateIp6Instance(), Ping6OnEchoReplyReceived(), Ping6OnTimerRoutine(), ShowNicVlanInfo(), and VlanConfigMain().

EFI_IPSEC_CONFIG_PROTOCOL* mIpSecConfig

Referenced by AddOrInsertPolicyEntry(), DeletePolicyEntry(), EditOperatePolicyEntry(), FlushOrDeletePolicyEntry(), ForeachPolicyEntry(), InitializeIpSecConfig(), and InsertPolicyEntry().

STR2INT mMapAuthAlgo[]

Referenced by CreateSadEntry(), CreateSpdEntry(), DumpSadEntry(), and DumpSpdEntry().

STR2INT mMapAuthMethod[]

Referenced by CreatePadEntry(), and DumpPadEntry().

STR2INT mMapAuthProto[]

Referenced by CreatePadEntry(), and DumpPadEntry().

STR2INT mMapDfOption[]

Referenced by CreateSpdEntry(), and DumpSpdEntry().

STR2INT mMapEncAlgo[]

Referenced by CreateSadEntry(), CreateSpdEntry(), DumpSadEntry(), and DumpSpdEntry().

STR2INT mMapIpProtocol[]

Referenced by CreateSpdSelector(), and DumpSpdSelector().

STR2INT mMapIpSecAction[]

Referenced by CreateSpdEntry(), and DumpSpdEntry().

STR2INT mMapIpSecMode[]

Referenced by CreateSadEntry(), CreateSpdEntry(), DumpSadEntry(), and DumpSpdEntry().

STR2INT mMapIpSecProtocol[]

Referenced by ConstructSadIndexer(), CreateSadEntry(), CreateSpdEntry(), DumpSadEntry(), and DumpSpdEntry().

STR2INT mMapPolicy[]

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Application/IpsecConfig/Match.c File Reference

Functions

BOOLEAN IsMemoryZero (IN VOID *Memory, IN UINTN Size)
BOOLEAN MatchSpdEntry (IN EFI_IPSEC_SPD_SELECTOR *Selector, IN EFI_IPSEC_SPD_DATA *Data, IN SPD_ENTRY_INDEXER *Indexer)
BOOLEAN MatchSadEntry (IN EFI_IPSEC_SA_ID *SaId, IN EFI_IPSEC_SA_DATA2 *Data, IN SAD_ENTRY_INDEXER *Indexer)
BOOLEAN MatchPadEntry (IN EFI_IPSEC_PAD_ID *PadId, IN EFI_IPSEC_PAD_DATA *Data, IN PAD_ENTRY_INDEXER *Indexer)

Variables

MATCH_POLICY_ENTRY mMatchPolicyEntry []

Detailed Description

The implementation of match policy entry function in IpSecConfig application.

Copyright (c) 2009 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
BOOLEAN IsMemoryZero ( IN VOID *   Memory,
                       IN UINTN    Size
                     )
```

Private function to validate a buffer that should be filled with zero.

Parameters:

[in]	*Memory*	The pointer to the buffer.
[in]	*Size*	The size of the buffer.

Return values:

TRUE	The memory is filled with zero.
FALSE	The memory isn't filled with zero.

Referenced by MatchPadEntry(), and MatchSadEntry().

```
BOOLEAN MatchPadEntry ( IN EFI_IPSEC_PAD_ID *        PadId,
                        IN EFI_IPSEC_PAD_DATA *      Data,
                        IN PAD_ENTRY_INDEXER *       Indexer
                      )
```

Find the matching PAD with Indexer.

Parameters:

[in] *PadId* The pointer to the EFI_IPSEC_PAD_ID structure.
[in] *Data* The pointer to the EFI_IPSEC_PAD_DATA structure.
[in] *Indexer* The pointer to the SPD_ENTRY_INDEXER structure.

Return values:

TRUE The matched PAD is found.
FALSE The matched PAD is not found.

References IsMemoryZero().

```
BOOLEAN MatchSadEntry ( IN EFI_IPSEC_SA_ID *         SaId,
                        IN EFI_IPSEC_SA_DATA2 *      Data,
                        IN SAD_ENTRY_INDEXER *       Indexer
                      )
```

Find the matching SAD with Indexer.

Parameters:

[in] *SaId* The pointer to the EFI_IPSEC_SA_ID structure.
[in] *Data* The pointer to the EFI_IPSEC_SA_DATA2 structure.
[in] *Indexer* The pointer to the SPD_ENTRY_INDEXER structure.

Return values:

TRUE The matched SAD is found.
FALSE The matched SAD is not found.

References IsMemoryZero().

```
BOOLEAN MatchSpdEntry ( IN EFI_IPSEC_SPD_SELECTOR *  Selector,
                        IN EFI_IPSEC_SPD_DATA *      Data,
                        IN SPD_ENTRY_INDEXER *       Indexer
                      )
```

Find the matching SPD with Indexer.

Parameters:

[in] *Selector* The pointer to the EFI_IPSEC_SPD_SELECTOR structure.
[in] *Data* The pointer to the EFI_IPSEC_SPD_DATA structure.
[in] *Indexer* The pointer to the SPD_ENTRY_INDEXER structure.

Return values:

TRUE The matched SPD is found.
FALSE The matched SPD is not found.

Variable Documentation

MATCH_POLICY_ENTRY mMatchPolicyEntry[]

Initial value:

```
{
  (MATCH_POLICY_ENTRY) MatchSpdEntry,
  (MATCH_POLICY_ENTRY) MatchSadEntry,
  (MATCH_POLICY_ENTRY) MatchPadEntry
}
```

Referenced by DeletePolicyEntry(), EditOperatePolicyEntry(), and InsertPolicyEntry().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Application/IpsecConfig/Match.h File Reference

Typedefs

typedef BOOLEAN(* MATCH_POLICY_ENTRY)(IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data, IN POLICY_ENTRY_INDEXER *Indexer)

Variables

MATCH_POLICY_ENTRY mMatchPolicyEntry []

Detailed Description

The internal structure and function declaration of match policy entry function in IpSecConfig application.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Typedef Documentation

typedef BOOLEAN(* MATCH_POLICY_ENTRY)(IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data, IN POLICY_ENTRY_INDEXER *Indexer)

The prototype for the MatchSpdEntry()/MatchSadEntry()/MatchPadEntry(). The functionality is to find the matching SPD/SAD/PAD with Indexer.

Parameters:

[in]	*Selector*	The pointer to the EFI_IPSEC_CONFIG_SELECTOR union.
[in]	*Data*	The pointer to corresponding Data.
[in]	*Indexer*	The pointer to the POLICY_ENTRY_INDEXER union.

Return values:

TRUE	The matched SPD/SAD/PAD is found.
FALSE	The matched SPD/SAD/PAD is not found.

Variable Documentation

MATCH_POLICY_ENTRY mMatchPolicyEntry[]

Referenced by DeletePolicyEntry(), EditOperatePolicyEntry(), and InsertPolicyEntry().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Application/IpsecConfig/PolicyEntryOpera
File Reference

Functions

EFI_STATUS CreateSpdSelector (OUT EFI_IPSEC_SPD_SELECTOR *Selector, IN LIST_ENTRY *ParamPackage, IN OUT UINT32 *Mask)
EFI_STATUS CreateSpdEntry (OUT EFI_IPSEC_SPD_SELECTOR **Selector, OUT EFI_IPSEC_SPD_DATA **Data, IN LIST_ENTRY *ParamPackage, OUT UINT32 *Mask, IN BOOLEAN CreateNew)
EFI_STATUS CreateSadEntry (OUT EFI_IPSEC_SA_ID **SaId, OUT EFI_IPSEC_SA_DATA2 **Data, IN LIST_ENTRY *ParamPackage, OUT UINT32 *Mask, IN BOOLEAN CreateNew)
EFI_STATUS CreatePadEntry (OUT EFI_IPSEC_PAD_ID **PadId, OUT EFI_IPSEC_PAD_DATA **Data, IN LIST_ENTRY *ParamPackage, OUT UINT32 *Mask, IN BOOLEAN CreateNew)
EFI_STATUS CombineSpdEntry (IN OUT EFI_IPSEC_SPD_SELECTOR *OldSelector, IN OUT EFI_IPSEC_SPD_DATA *OldData, IN EFI_IPSEC_SPD_SELECTOR *NewSelector, IN EFI_IPSEC_SPD_DATA *NewData, IN UINT32 Mask, OUT BOOLEAN *CreateNew)
EFI_STATUS CombineSadEntry (IN OUT EFI_IPSEC_SA_ID *OldSaId, IN OUT EFI_IPSEC_SA_DATA2 *OldData, IN EFI_IPSEC_SA_ID *NewSaId, IN EFI_IPSEC_SA_DATA2 *NewData, IN UINT32 Mask, OUT BOOLEAN *CreateNew)
EFI_STATUS CombinePadEntry (IN OUT EFI_IPSEC_PAD_ID *OldPadId, IN OUT EFI_IPSEC_PAD_DATA *OldData, IN EFI_IPSEC_PAD_ID *NewPadId, IN EFI_IPSEC_PAD_DATA *NewData, IN UINT32 Mask, OUT BOOLEAN *CreateNew)
EFI_STATUS EditOperatePolicyEntry (IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data, IN EDIT_POLICY_ENTRY_CONTEXT *Context)
EFI_STATUS EditPolicyEntry (IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN LIST_ENTRY *ParamPackage)
EFI_STATUS InsertPolicyEntry (IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data, IN INSERT_POLICY_ENTRY_CONTEXT *Context)
EFI_STATUS AddOrInsertPolicyEntry (IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN LIST_ENTRY *ParamPackage)

Variables

CREATE_POLICY_ENTRY mCreatePolicyEntry []
COMBINE_POLICY_ENTRY mCombinePolicyEntry []

Detailed Description

The implementation of policy entry operation function in IpSecConfig application.

Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS AddOrInsertPolicyEntry ( IN EFI_IPSEC_CONFIG_DATA_TYPE  DataType,
                                    IN LIST_ENTRY *                ParamPackage
                                  )
```

Insert or add entry information in database according to datatype.

Parameters:

[in]	*DataType*	The value of EFI_IPSEC_CONFIG_DATA_TYPE.
[in]	*ParamPackage*	The pointer to the ParamPackage list.

Return values:

EFI_SUCCESS	Insert or add entry information successfully.
EFI_NOT_FOUND	Can't find the specified entry.
EFI_BUFFER_TOO_SMALL	The entry already existed.
EFI_UNSUPPORTED	The operation is not supported.
Others	Some mistaken case.

References INSERT_POLICY_ENTRY_CONTEXT::Data, INSERT_POLICY_ENTRY_CONTEXT::DataType, ForeachPolicyEntry(), INSERT_POLICY_ENTRY_CONTEXT::Indexer, InsertPolicyEntry(), mAppName, mConstructPolicyEntryIndexer, mCreatePolicyEntry, mHiiHandle, mIpSecConfig, INSERT_POLICY_ENTRY_CONTEXT::Selector, and INSERT_POLICY_ENTRY_CONTEXT::Status.

Referenced by InitializeIpSecConfig().

```
EFI_STATUS CombinePadEntry ( IN OUT EFI_IPSEC_PAD_ID *        OldPadId,
                             IN OUT EFI_IPSEC_PAD_DATA *      OldData,
                             IN EFI_IPSEC_PAD_ID *            NewPadId,
                             IN EFI_IPSEC_PAD_DATA *          NewData,
                             IN UINT32                        Mask,
                             OUT BOOLEAN *                    CreateNew
                           )
```

Combine old PAD entry with new PAD entry.

Parameters:

[in,out]	*OldPadId*	The pointer to the EFI_IPSEC_PAD_ID structure.
[in,out]	*OldData*	The pointer to the EFI_IPSEC_PAD_DATA structure.
[in]	*NewPadId*	The pointer to the EFI_IPSEC_PAD_ID structure.
[in]	*NewData*	The pointer to the EFI_IPSEC_PAD_DATA structure.
[in]	*Mask*	The pointer to the Mask.
[out]	*CreateNew*	The switch to create new.

Return values:

EFI_SUCCESS	Combined successfully.
EFI_INVALID_PARAMETER	Invalid user input parameter.

References AUTH_DATA, AUTH_METHOD, AUTH_PROTO, IKE_ID, PEER_ADDRESS, PEER_ID, and REVOCATION_DATA.

```
EFI_STATUS CombineSadEntry ( IN OUT EFI_IPSEC_SA_ID *          OldSaId,
                             IN OUT EFI_IPSEC_SA_DATA2 *       OldData,
                             IN EFI_IPSEC_SA_ID *              NewSaId,
                             IN EFI_IPSEC_SA_DATA2 *           NewData,
                             IN UINT32                         Mask,
                             OUT BOOLEAN *                     CreateNew
                           )
```

Combine old SAD entry with new SAD entry.

Parameters:

`[in,out]`	*OldSaId*	The pointer to the EFI_IPSEC_SA_ID structure.
`[in,out]`	*OldData*	The pointer to the EFI_IPSEC_SA_DATA2 structure.
`[in]`	*NewSaId*	The pointer to the EFI_IPSEC_SA_ID structure.
`[in]`	*NewData*	The pointer to the EFI_IPSEC_SA_DATA2 structure.
`[in]`	*Mask*	The pointer to the Mask.
`[out]`	*CreateNew*	The switch to create new.

Return values:

EFI_SUCCESS	Combined successfully.
EFI_INVALID_PARAMETER	Invalid user input parameter.

References ANTIREPLAY_WINDOW, AUTH_ALGO, AUTH_KEY, DEST, EFI_IP4_PROTO_ICMP, EFI_IP4_PROTO_TCP, EFI_IP4_PROTO_UDP, ENCRYPT_ALGO, ENCRYPT_KEY, ICMP_CODE, ICMP_TYPE, IPSEC_EALG_NONE, IPSEC_PROTO, LIFEBYTE, LIFETIME, LIFETIME_SOFT, LOCAL, LOCAL_PORT, mAppName, mHiiHandle, MODE, PATH_MTU, PROTO, REMOTE, REMOTE_PORT, SEQUENCE_NUMBER, SOURCE, and SPI.

```
EFI_STATUS CombineSpdEntry ( IN OUT EFI_IPSEC_SPD_SELECTOR *   OldSelector,
                             IN OUT EFI_IPSEC_SPD_DATA *       OldData,
                             IN EFI_IPSEC_SPD_SELECTOR *       NewSelector,
                             IN EFI_IPSEC_SPD_DATA *           NewData,
                             IN UINT32                         Mask,
                             OUT BOOLEAN *                     CreateNew
                           )
```

Combine old SPD entry with new SPD entry.

Parameters:

`[in,out]`	*OldSelector*	The pointer to the EFI_IPSEC_SPD_SELECTOR structure.
`[in,out]`	*OldData*	The pointer to the EFI_IPSEC_SPD_DATA structure.
`[in]`	*NewSelector*	The pointer to the EFI_IPSEC_SPD_SELECTOR structure.
`[in]`	*NewData*	The pointer to the EFI_IPSEC_SPD_DATA structure.

[in]	*Mask*	The pointer to the Mask.
[out]	*CreateNew*	The switch to create new.

Return values:

EFI_SUCCESS	Combined successfully.
EFI_INVALID_PARAMETER	Invalid user input parameter.

References ACTION, AUTH_ALGO, DONT_FRAGMENT, EFI_IP4_PROTO_ICMP, EFI_IP4_PROTO_TCP, EFI_IP4_PROTO_UDP, ENCRYPT_ALGO, EXT_SEQUENCE, FRAGMENT_CHECK, ICMP_CODE, ICMP_TYPE, IPSEC_PROTO, LIFEBYTE, LIFETIME, LIFETIME_SOFT, LOCAL, LOCAL_PORT, mAppName, mHiiHandle, MODE, NAME, PACKET_FLAG, PROTO, REMOTE, REMOTE_PORT, SEQUENCE_OVERFLOW, TUNNEL_LOCAL, and TUNNEL_REMOTE.

```
EFI_STATUS CreatePadEntry ( OUT EFI_IPSEC_PAD_ID **        PadId,
                            OUT EFI_IPSEC_PAD_DATA **      Data,
                            IN LIST_ENTRY *                ParamPackage,
                            OUT UINT32 *                   Mask,
                            IN BOOLEAN                     CreateNew
                          )
```

Fill in EFI_IPSEC_PAD_ID and EFI_IPSEC_PAD_DATA through ParamPackage list.

Parameters:

[out]	*PadId*	The pointer to the EFI_IPSEC_PAD_ID structure.
[out]	*Data*	The pointer to the EFI_IPSEC_PAD_DATA structure.
[in]	*ParamPackage*	The pointer to the ParamPackage list.
[out]	*Mask*	The pointer to the Mask.
[in]	*CreateNew*	The switch to create new.

Return values:

EFI_SUCCESS	Fill in EFI_IPSEC_PAD_ID and EFI_IPSEC_PAD_DATA successfully.
EFI_INVALID_PARAMETER	Invalid user input parameter.

References AUTH_DATA, AUTH_METHOD, AUTH_PROTO, EfiInetAddrRange(), FORMAT_STRING, GetNumber(), IKE_ID, mAppName, mHiiHandle, mMapAuthMethod, mMapAuthProto, PEER_ADDRESS, PEER_ID, and REVOCATION_DATA.

```
EFI_STATUS CreateSadEntry ( OUT EFI_IPSEC_SA_ID **         SaId,
                            OUT EFI_IPSEC_SA_DATA2 **      Data,
                            IN LIST_ENTRY *                ParamPackage,
                            OUT UINT32 *                   Mask,
                            IN BOOLEAN                     CreateNew
                          )
```

Fill in EFI_IPSEC_SA_ID and EFI_IPSEC_SA_DATA2 through ParamPackage list.

Parameters:

[out]	*SaId*	The pointer to the EFI_IPSEC_SA_ID structure.
[out]	*Data*	The pointer to the EFI_IPSEC_SA_DATA2 structure.

[in]	*ParamPackage*	The pointer to the ParamPackage list.
[out]	*Mask*	The pointer to the Mask.
[in]	*CreateNew*	The switch to create new.

Return values:

EFI_SUCCESS	Fill in EFI_IPSEC_SA_ID and EFI_IPSEC_SA_DATA2 successfully.
EFI_INVALID_PARAMETER	Invalid user input parameter.

References AUTH_ALGO, AUTH_KEY, CreateSpdSelector(), DEST, EfiInetAddr2(), ENCRYPT_ALGO, ENCRYPT_KEY, FORMAT_NUMBER, FORMAT_STRING, GetNumber(), IPSEC_AALG_NONE, IPSEC_EALG_NONE, IPSEC_PROTO, LIFEBYTE, LIFETIME, LIFETIME_SOFT, LOCAL, mAppName, mHiiHandle, mMapAuthAlgo, mMapEncAlgo, mMapIpSecMode, mMapIpSecProtocol, MODE, PATH_MTU, REMOTE, SEQUENCE_NUMBER, SOURCE, and SPI.

```
EFI_STATUS CreateSpdEntry ( OUT EFI_IPSEC_SPD_SELECTOR **  Selector,
                            OUT EFI_IPSEC_SPD_DATA **      Data,
                            IN LIST_ENTRY *                ParamPackage,
                            OUT UINT32 *                   Mask,
                            IN BOOLEAN                     CreateNew
                          )
```

Fill in EFI_IPSEC_SPD_SELECTOR and EFI_IPSEC_SPD_DATA through ParamPackage list.

Parameters:

[out]	*Selector*	The pointer to the EFI_IPSEC_SPD_SELECTOR structure.
[out]	*Data*	The pointer to the EFI_IPSEC_SPD_DATA structure.
[in]	*ParamPackage*	The pointer to the ParamPackage list.
[out]	*Mask*	The pointer to the Mask.
[in]	*CreateNew*	The switch to create new.

Return values:

EFI_SUCCESS	Fill in EFI_IPSEC_SPD_SELECTOR and EFI_IPSEC_SPD_DATA successfully.
EFI_INVALID_PARAMETER	Invalid user input parameter.

References ACTION, AUTH_ALGO, CreateSpdSelector(), DONT_FRAGMENT, EfiInetAddr2(), ENCRYPT_ALGO, EXT_SEQUENCE, FORMAT_NUMBER, FORMAT_STRING, FRAGMENT_CHECK, GetNumber(), IPSEC_PROTO, LIFEBYTE, LIFETIME, LIFETIME_SOFT, LOCAL, mAppName, mHiiHandle, mMapAuthAlgo, mMapDfOption, mMapEncAlgo, mMapIpSecAction, mMapIpSecMode, mMapIpSecProtocol, MODE, NAME, PACKET_FLAG, PROTO, REMOTE, SEQUENCE_OVERFLOW, TUNNEL_LOCAL, and TUNNEL_REMOTE.

```
EFI_STATUS CreateSpdSelector ( OUT EFI_IPSEC_SPD_SELECTOR *  Selector,
                               IN LIST_ENTRY *               ParamPackage,
                               IN OUT UINT32 *               Mask
                             )
```

Fill in EFI_IPSEC_SPD_SELECTOR through ParamPackage list.

Parameters:

[out]	*Selector*	The pointer to the EFI_IPSEC_SPD_SELECTOR structure.
[in]	*ParamPackage*	The pointer to the ParamPackage list.
[in,out]	*Mask*	The pointer to the Mask.

Return values:

EFI_SUCCESS	Fill in EFI_IPSEC_SPD_SELECTOR successfully.
EFI_INVALID_PARAMETER	Invalid user input parameter.

References EFI_IPSEC_ANY_PORT, EFI_IPSEC_ANY_PROTOCOL, EfiInetAddrRange(), EfiInetPortRange(), FORMAT_NUMBER, FORMAT_STRING, GetNumber(), ICMP_CODE, ICMP_TYPE, LOCAL, LOCAL_PORT, mAppName, mHiiHandle, mMapIpProtocol, PROTO, REMOTE, and REMOTE_PORT.

Referenced by CreateSadEntry(), and CreateSpdEntry().

```
EFI_STATUS EditOperatePolicyEntry ( IN EFI_IPSEC_CONFIG_SELECTOR *      Selector,
                                    IN VOID *                           Data,
                                    IN EDIT_POLICY_ENTRY_CONTEXT *      Context
                                  )
```

Edit entry information in the database.

Parameters:

[in]	*Selector*	The pointer to the EFI_IPSEC_CONFIG_SELECTOR structure.
[in]	*Data*	The pointer to the data.
[in]	*Context*	The pointer to the INSERT_POLICY_ENTRY_CONTEXT structure.

Return values:

EFI_SUCCESS	Continue the iteration.
EFI_ABORTED	Abort the iteration.

References mIpSecConfig, and mMatchPolicyEntry.

Referenced by EditPolicyEntry().

```
EFI_STATUS EditPolicyEntry ( IN EFI_IPSEC_CONFIG_DATA_TYPE  DataType,
                             IN LIST_ENTRY *                ParamPackage
                           )
```

Edit entry information in database according to datatype.

Parameters:

[in]	*DataType*	The value of EFI_IPSEC_CONFIG_DATA_TYPE.
[in]	*ParamPackage*	The pointer to the ParamPackage list.

Return values:

EFI_SUCCESS	Edit entry information successfully.
EFI_NOT_FOUND	Can't find the specified entry.
Others	Some mistaken case.

References EDIT_POLICY_ENTRY_CONTEXT::Data, EDIT_POLICY_ENTRY_CONTEXT::DataType, EditOperatePolicyEntry(), ForeachPolicyEntry(), EDIT_POLICY_ENTRY_CONTEXT::Indexer,

mAppName, EDIT_POLICY_ENTRY_CONTEXT::Mask, mConstructPolicyEntryIndexer, mCreatePolicyEntry, mHiiHandle, EDIT_POLICY_ENTRY_CONTEXT::Selector, and EDIT_POLICY_ENTRY_CONTEXT::Status.

Referenced by InitializeIpSecConfig().

```
EFI_STATUS InsertPolicyEntry ( IN EFI_IPSEC_CONFIG_SELECTOR *        Selector,
                               IN VOID *                             Data,
                               IN INSERT_POLICY_ENTRY_CONTEXT *      Context
                             )
```

Insert entry information in database.

Parameters:

[in]	*Selector*	The pointer to the EFI_IPSEC_CONFIG_SELECTOR structure.
[in]	*Data*	The pointer to the data.
[in]	*Context*	The pointer to the INSERT_POLICY_ENTRY_CONTEXT structure.

Return values:

EFI_SUCCESS	Continue the iteration.
EFI_ABORTED	Abort the iteration.

References mIpSecConfig, and mMatchPolicyEntry.

Referenced by AddOrInsertPolicyEntry().

Variable Documentation

COMBINE_POLICY_ENTRY mCombinePolicyEntry[]

Initial value:

```
{
  (COMBINE_POLICY_ENTRY) CombineSpdEntry,
  (COMBINE_POLICY_ENTRY) CombineSadEntry,
  (COMBINE_POLICY_ENTRY) CombinePadEntry
}
```

CREATE_POLICY_ENTRY mCreatePolicyEntry[]

Initial value:

```
{
  (CREATE_POLICY_ENTRY) CreateSpdEntry,
  (CREATE_POLICY_ENTRY) CreateSadEntry,
  (CREATE_POLICY_ENTRY) CreatePadEntry
}
```

Referenced by AddOrInsertPolicyEntry(), and EditPolicyEntry().

- File List
- Globals

NetworkPkg/Application/IpsecConfig/PolicyEntryOpera
File Reference

Data Structures

struct EDIT_POLICY_ENTRY_CONTEXT
struct INSERT_POLICY_ENTRY_CONTEXT

Defines

"memItemRight" valign="bottom">LOCAL BIT(0)
#define REMOTE BIT(1)
#define PROTO BIT(2)
#define LOCAL_PORT BIT(3)
#define REMOTE_PORT BIT(4)
#define ICMP_TYPE BIT(5)
#define ICMP_CODE BIT(6)
#define NAME BIT(7)
#define PACKET_FLAG BIT(8)
#define ACTION BIT(9)
#define EXT_SEQUENCE BIT(10)
#define SEQUENCE_OVERFLOW BIT(11)
#define FRAGMENT_CHECK BIT(12)
#define LIFEBYTE BIT(13)
#define LIFETIME_SOFT BIT(14)
#define LIFETIME BIT(15)
#define MODE BIT(16)
#define TUNNEL_LOCAL BIT(17)
#define TUNNEL_REMOTE BIT(18)
#define DONT_FRAGMENT BIT(19)
#define IPSEC_PROTO BIT(20)
#define AUTH_ALGO BIT(21)
#define ENCRYPT_ALGO BIT(22)
#define SPI BIT(23)
#define DEST BIT(24)
#define SEQUENCE_NUMBER BIT(25)
#define ANTIREPLAY_WINDOW BIT(26)
#define AUTH_KEY BIT(27)
#define ENCRYPT_KEY BIT(28)
#define PATH_MTU BIT(29)
#define SOURCE BIT(30)
#define PEER_ID BIT(0)
#define PEER_ADDRESS BIT(1)
#define AUTH_PROTO BIT(2)
#define AUTH_METHOD BIT(3)
#define IKE_ID BIT(4)
#define AUTH_DATA BIT(5)
#define REVOCATION_DATA BIT(6)

Typedefs

typedef EFI_STATUS(* CREATE_POLICY_ENTRY)(OUT EFI_IPSEC_CONFIG_SELECTOR **Selector, OUT VOID **Data, IN LIST_ENTRY *ParamPackage, OUT UINT32 *Mask, IN BOOLEAN CreateNew)

typedef EFI_STATUS(* COMBINE_POLICY_ENTRY)(IN OUT EFI_IPSEC_CONFIG_SELECTOR *OldSelector, IN OUT VOID *OldData, IN EFI_IPSEC_CONFIG_SELECTOR *NewSelector, IN VOID *NewData, IN UINT32 Mask, OUT BOOLEAN *CreateNew)

Functions

EFI_STATUS AddOrInsertPolicyEntry (IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN LIST_ENTRY *ParamPackage)

EFI_STATUS EditPolicyEntry (IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN LIST_ENTRY *ParamPackage)

Detailed Description

The function declaration of policy entry operation in IpSecConfig application.

Copyright (c) 2009 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by CombineSpdEntry(), and CreateSpdEntry().

#define ANTIREPLAY_WINDOW BIT(26)

Referenced by CombineSadEntry().

#define AUTH_ALGO BIT(21)

Referenced by CombineSadEntry(), CombineSpdEntry(), CreateSadEntry(), and CreateSpdEntry().

#define AUTH_DATA BIT(5)

Referenced by CombinePadEntry(), and CreatePadEntry().

#define AUTH_KEY BIT(27)

Referenced by CombineSadEntry(), and CreateSadEntry().

#define AUTH_METHOD BIT(3)

Referenced by CombinePadEntry(), and CreatePadEntry().

#define AUTH_PROTO BIT(2)

Referenced by CombinePadEntry(), and CreatePadEntry().

#define DEST BIT(24)

Referenced by CombineSadEntry(), and CreateSadEntry().

#define DONT_FRAGMENT BIT(19)

Referenced by CombineSpdEntry(), and CreateSpdEntry().

#define ENCRYPT_ALGO BIT(22)

Referenced by CombineSadEntry(), CombineSpdEntry(), CreateSadEntry(), and CreateSpdEntry().

#define ENCRYPT_KEY BIT(28)

Referenced by CombineSadEntry(), and CreateSadEntry().

#define EXT_SEQUENCE BIT(10)

Referenced by CombineSpdEntry(), and CreateSpdEntry().

#define FRAGMENT_CHECK BIT(12)

Referenced by CombineSpdEntry(), and CreateSpdEntry().

#define ICMP_CODE BIT(6)

Referenced by CombineSadEntry(), CombineSpdEntry(), and CreateSpdSelector().

#define ICMP_TYPE BIT(5)

Referenced by CombineSadEntry(), CombineSpdEntry(), and CreateSpdSelector().

#define IKE_ID BIT(4)

Referenced by CombinePadEntry(), and CreatePadEntry().

#define IPSEC_PROTO BIT(20)

Referenced by CombineSadEntry(), CombineSpdEntry(), CreateSadEntry(), and CreateSpdEntry().

#define LIFEBYTE BIT(13)

Referenced by CombineSadEntry(), CombineSpdEntry(), CreateSadEntry(), and CreateSpdEntry().

#define LIFETIME BIT(15)

Referenced by CombineSadEntry(), CombineSpdEntry(), CreateSadEntry(), and CreateSpdEntry().

#define LIFETIME_SOFT BIT(14)

Referenced by CombineSadEntry(), CombineSpdEntry(), CreateSadEntry(), and CreateSpdEntry().

#define LOCAL BIT(0)

Referenced by CombineSadEntry(), CombineSpdEntry(), CreateSadEntry(), CreateSpdEntry(), and CreateSpdSelector().

#define LOCAL_PORT BIT(3)

Referenced by CombineSadEntry(), CombineSpdEntry(), and CreateSpdSelector().

#define MODE BIT(16)

Referenced by CombineSadEntry(), CombineSpdEntry(), CreateSadEntry(), and CreateSpdEntry().

#define NAME BIT(7)

Referenced by CombineSpdEntry(), and CreateSpdEntry().

#define PACKET_FLAG BIT(8)

Referenced by CombineSpdEntry(), and CreateSpdEntry().

#define PATH_MTU BIT(29)

Referenced by CombineSadEntry(), and CreateSadEntry().

#define PEER_ADDRESS BIT(1)

Referenced by CombinePadEntry(), and CreatePadEntry().

#define PEER_ID BIT(0)

Referenced by CombinePadEntry(), and CreatePadEntry().

#define PROTO BIT(2)

Referenced by CombineSadEntry(), CombineSpdEntry(), CreateSpdEntry(), and CreateSpdSelector().

#define REMOTE BIT(1)

Referenced by CombineSadEntry(), CombineSpdEntry(), CreateSadEntry(), CreateSpdEntry(), and CreateSpdSelector().

#define REMOTE_PORT BIT(4)

Referenced by CombineSadEntry(), CombineSpdEntry(), and CreateSpdSelector().

#define REVOCATION_DATA BIT(6)

Referenced by CombinePadEntry(), and CreatePadEntry().

#define SEQUENCE_NUMBER BIT(25)

Referenced by CombineSadEntry(), and CreateSadEntry().

#define SEQUENCE_OVERFLOW BIT(11)

Referenced by CombineSpdEntry(), and CreateSpdEntry().

#define SOURCE BIT(30)

Referenced by CombineSadEntry(), and CreateSadEntry().

#define SPI BIT(23)

Referenced by CombineSadEntry(), and CreateSadEntry().

#define TUNNEL_LOCAL BIT(17)

Referenced by CombineSpdEntry(), and CreateSpdEntry().

#define TUNNEL_REMOTE BIT(18)

Referenced by CombineSpdEntry(), and CreateSpdEntry().

Typedef Documentation

typedef EFI_STATUS(* COMBINE_POLICY_ENTRY)(IN OUT EFI_IPSEC_CONFIG_SELECTOR *OldSelector, IN OUT VOID *OldData, IN EFI_IPSEC_CONFIG_SELECTOR *NewSelector, IN VOID *NewData, IN UINT32 Mask, OUT BOOLEAN *CreateNew)

The prototype for the CombineSpdEntry()/CombineSadEntry()/CombinePadEntry(). Combine old SPD/SAD/PAD entry with new SPD/SAD/PAD entry.

Parameters:

[in,out]	*OldSelector*	The pointer to the old EFI_IPSEC_CONFIG_SELECTOR union.
[in,out]	*OldData*	The pointer to the corresponding old data.
[in]	*NewSelector*	The pointer to the new EFI_IPSEC_CONFIG_SELECTOR union.
[in]	*NewData*	The pointer to the corresponding new data.
[in]	*Mask*	The pointer to the Mask.
[out]	*CreateNew*	The switch to create new.

Return values:

EFI_SUCCESS	Combined successfully.
EFI_INVALID_PARAMETER	Invalid user input parameter.

typedef EFI_STATUS(* CREATE_POLICY_ENTRY)(OUT EFI_IPSEC_CONFIG_SELECTOR **Selector, OUT VOID **Data, IN LIST_ENTRY *ParamPackage, OUT UINT32 *Mask, IN BOOLEAN CreateNew)

The prototype for the CreateSpdEntry()/CreateSadEntry()/CreatePadEntry(). Fill in EFI_IPSEC_CONFIG_SELECTOR and corresponding data thru ParamPackage list.

Parameters:

[out]	*Selector*	The pointer to the EFI_IPSEC_CONFIG_SELECTOR union.
[out]	*Data*	The pointer to corresponding data.
[in]	*ParamPackage*	The pointer to the ParamPackage list.
[out]	*Mask*	The pointer to the Mask.
[in]	*CreateNew*	The switch to create new.

Return values:

EFI_SUCCESS	Filled in EFI_IPSEC_CONFIG_SELECTOR and corresponding data successfully.
EFI_INVALID_PARAMETER	Invalid user input parameter.

Function Documentation

```
EFI_STATUS AddOrInsertPolicyEntry ( IN EFI_IPSEC_CONFIG_DATA_TYPE  DataType,
                                    IN LIST_ENTRY *                ParamPackage
                                  )
```

Insert or add entry information in database according to datatype.

Parameters:

`[in]`	*DataType*	The value of EFI_IPSEC_CONFIG_DATA_TYPE.
`[in]`	*ParamPackage*	The pointer to the ParamPackage list.

Return values:

EFI_SUCCESS	Insert or add entry information successfully.
EFI_NOT_FOUND	Can't find the specified entry.
EFI_BUFFER_TOO_SMALL	The entry already existed.
EFI_UNSUPPORTED	The operation is not supported./
Others	Some mistaken case.

Insert or add entry information in database according to datatype.

Parameters:

`[in]`	*DataType*	The value of EFI_IPSEC_CONFIG_DATA_TYPE.
`[in]`	*ParamPackage*	The pointer to the ParamPackage list.

Return values:

EFI_SUCCESS	Insert or add entry information successfully.
EFI_NOT_FOUND	Can't find the specified entry.
EFI_BUFFER_TOO_SMALL	The entry already existed.
EFI_UNSUPPORTED	The operation is not supported.
Others	Some mistaken case.

References INSERT_POLICY_ENTRY_CONTEXT::Data, INSERT_POLICY_ENTRY_CONTEXT::DataType, ForeachPolicyEntry(), INSERT_POLICY_ENTRY_CONTEXT::Indexer, InsertPolicyEntry(), mAppName, mConstructPolicyEntryIndexer, mCreatePolicyEntry, mHiiHandle, mIpSecConfig, INSERT_POLICY_ENTRY_CONTEXT::Selector, and INSERT_POLICY_ENTRY_CONTEXT::Status.

Referenced by InitializeIpSecConfig().

```
EFI_STATUS EditPolicyEntry ( IN EFI_IPSEC_CONFIG_DATA_TYPE  DataType,
                             IN LIST_ENTRY *                ParamPackage
                           )
```

Edit entry information in the database according to datatype.

Parameters:

[in]	*DataType*	The value of EFI_IPSEC_CONFIG_DATA_TYPE.
[in]	*ParamPackage*	The pointer to the ParamPackage list.

Return values:

EFI_SUCCESS	Edit entry information successfully.
EFI_NOT_FOUND	Can't find the specified entry.
Others	Some mistaken case.

Edit entry information in database according to datatype.

Parameters:

[in]	*DataType*	The value of EFI_IPSEC_CONFIG_DATA_TYPE.
[in]	*ParamPackage*	The pointer to the ParamPackage list.

Return values:

EFI_SUCCESS	Edit entry information successfully.
EFI_NOT_FOUND	Can't find the specified entry.
Others	Some mistaken case.

References EDIT_POLICY_ENTRY_CONTEXT::Data, EDIT_POLICY_ENTRY_CONTEXT::DataType, EditOperatePolicyEntry(), ForeachPolicyEntry(), EDIT_POLICY_ENTRY_CONTEXT::Indexer, mAppName, EDIT_POLICY_ENTRY_CONTEXT::Mask, mConstructPolicyEntryIndexer, mCreatePolicyEntry, mHiiHandle, EDIT_POLICY_ENTRY_CONTEXT::Selector, and EDIT_POLICY_ENTRY_CONTEXT::Status.

Referenced by InitializeIpSecConfig().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Application/Ping6/Ping6.c File Reference

Functions

EFI_STATUS Ping6GetFrequency (VOID)
UINT64 Ping6CalculateTick (IN UINT64 Begin, IN UINT64 End)
VOID Ping6DestroyTxInfo (IN PING6_ICMP6_TX_INFO *TxInfo)
EFI_STATUS Ping6MatchEchoReply (IN PING6_PRIVATE_DATA *Private, IN ICMP6_ECHO_REQUEST_REPLY *Packet)
VOID EFIAPI Ping6OnEchoRequestSent (IN EFI_EVENT Event, IN VOID *Context)
VOID EFIAPI Ping6OnEchoReplyReceived (IN EFI_EVENT Event, IN VOID *Context)
EFI_IP6_COMPLETION_TOKEN * Ping6GenerateToken (IN PING6_PRIVATE_DATA *Private, IN UINT64 TimeStamp, IN UINT16 SequenceNum)
EFI_STATUS Ping6SendEchoRequest (IN PING6_PRIVATE_DATA *Private)
EFI_STATUS Ping6ReceiveEchoReply (IN PING6_PRIVATE_DATA *Private)
VOID EFIAPI Ping6OnTimerRoutine (IN EFI_EVENT Event, IN VOID *Context)
EFI_STATUS Ping6CreateIp6Instance (IN PING6_PRIVATE_DATA *Private)
VOID Ping6DestroyIp6Instance (IN PING6_PRIVATE_DATA *Private)
EFI_STATUS Ping6 (IN EFI_HANDLE ImageHandle, IN UINT32 SendNumber, IN UINT32 BufferSize, IN EFI_IPv6_ADDRESS *SrcAddress, IN EFI_IPv6_ADDRESS *DstAddress)
EFI_STATUS EFIAPI InitializePing6 (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)

Variables

SHELL_PARAM_ITEM Ping6ParamList []
EFI_HII_HANDLE mHiiHandle
CONST CHAR16 * mIp6DstString
CONST CHAR16 * mIp6SrcString
UINT64 mFrequency = 0

Detailed Description

The implementation for Ping6 application.

Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI InitializePing6 ( IN EFI_HANDLE          ImageHandle,
                                    IN EFI_SYSTEM_TABLE *  SystemTable
                                  )
```

This is the declaration of an EFI image entry point. This entry point is the same for UEFI Applications, UEFI OS Loaders, and UEFI Drivers, including both device drivers and bus drivers.

The entry point for the Ping6 application that parses the command line input and calls the Ping6 process.

Parameters:

[in]	*ImageHandle*	The firmware allocated handle for the UEFI image.
[in]	*SystemTable*	A pointer to the EFI System Table.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETETR	Input parameters combination is invalid.
Others	Some errors occur.

References mHiiHandle, mIp6DstString, mIp6SrcString, Ping6(), PING6_MAX_BUFFER_SIZE, PING6_MAX_SEND_NUMBER, Ping6GetFrequency(), and Ping6ParamList.

```
EFI_STATUS Ping6 ( IN EFI_HANDLE          ImageHandle,
                   IN UINT32              SendNumber,
                   IN UINT32              BufferSize,
                   IN EFI_IPv6_ADDRESS *  SrcAddress,
                   IN EFI_IPv6_ADDRESS *  DstAddress
                 )
```

The Ping6 Process.

Parameters:

[in]	*ImageHandle*	The firmware allocated handle for the UEFI image.
[in]	*SendNumber*	The send request count.
[in]	*BufferSize*	The send buffer size.
[in]	*SrcAddress*	The source IPv6 address.
[in]	*DstAddress*	The destination IPv6 address.

Return values:

EFI_SUCCESS	The ping6 processed successfullly.
others	The ping6 processed unsuccessfully.

References _PING6_PRIVATE_DATA::BufferSize, _PING6_PRIVATE_DATA::DstAddress, _PING6_PRIVATE_DATA::ImageHandle, _PING6_PRIVATE_DATA::Ip6, _PING6_PRIVATE_DATA::Ip6ChildHandle, _PING6_ICMP6_TX_INFO::Link, mHiiHandle, mIp6DstString, PING6_ONE_SECOND, Ping6CreateIp6Instance(), Ping6DestroyIp6Instance(), Ping6DestroyTxInfo(), Ping6OnTimerRoutine(), Ping6ReceiveEchoReply(), Ping6SendEchoRequest(), _PING6_PRIVATE_DATA::RttMax, _PING6_PRIVATE_DATA::RttMin, _PING6_PRIVATE_DATA::RttSum, _PING6_PRIVATE_DATA::RxCount, _PING6_PRIVATE_DATA::RxToken, _PING6_PRIVATE_DATA::SendNum,

_PING6_PRIVATE_DATA::SrcAddress, _PING6_PRIVATE_DATA::Status, _PING6_PRIVATE_DATA::Timer, _PING6_ICMP6_TX_INFO::Token, _PING6_PRIVATE_DATA::TxCount, and _PING6_PRIVATE_DATA::TxList.

Referenced by InitializePing6().

UINT64 Ping6CalculateTick (IN UINT64 *Begin*,
IN UINT64 *End*
)

Get and calculate the duration in ms.

Parameters:

[in] *Begin* The start point of time.
[in] *End* The end point of time.

Returns:

The duration in ms.

References mFrequency.

Referenced by Ping6OnEchoReplyReceived(), and Ping6OnTimerRoutine().

EFI_STATUS Ping6CreateIp6Instance (IN PING6_PRIVATE_DATA * *Private*)

Create a valid IP6 instance.

Parameters:

[in] *Private* The pointer of PING6_PRIVATE_DATA.

Return values:

EFI_SUCCESS	Create a valid IP6 instance successfully.
EFI_ABORTED	Locate handle with ip6 service binding protocol unsuccessfully.
EFI_INVALID_PARAMETER	The source address is unspecified when the destination address is a link -ocal address.
EFI_OUT_OF_RESOURCES	No memory is available on the platform.
EFI_NOT_FOUND	The source address is not found.

References mHiiHandle, and mIp6SrcString.

Referenced by Ping6().

VOID Ping6DestroyIp6Instance (IN PING6_PRIVATE_DATA * *Private*)

Destroy the IP6 instance.

Parameters:

[in] *Private* The pointer of PING6_PRIVATE_DATA.

Referenced by Ping6().

VOID Ping6DestroyTxInfo (IN PING6_ICMP6_TX_INFO * *TxInfo*)

Destroy IPING6_ICMP6_TX_INFO, and recollect the memory.

Parameters:

[in] *TxInfo* The pointer to PING6_ICMP6_TX_INFO.

Referenced by Ping6(), Ping6MatchEchoReply(), Ping6OnTimerRoutine(), and Ping6SendEchoRequest().

```
EFI_IP6_COMPLETION_TOKEN*
Ping6GenerateToken                    ( IN PING6_PRIVATE_DATA *  Private,
                                        IN UINT64                TimeStamp,
                                        IN UINT16                SequenceNum
                                      )
```

Initial EFI_IP6_COMPLETION_TOKEN.

Parameters:

[in] *Private* The pointer of PING6_PRIVATE_DATA.
[in] *TimeStamp* The TimeStamp of request.
[in] *SequenceNum* The SequenceNum of request.

Returns:

The pointer of EFI_IP6_COMPLETION_TOKEN.

References _ICMP6_ECHO_REQUEST_REPLY::Checksum, _ICMP6_ECHO_REQUEST_REPLY::Code, _ICMP6_ECHO_REQUEST_REPLY::Identifier, Ping6OnEchoRequestSent(), _ICMP6_ECHO_REQUEST_REPLY::SequenceNum, _ICMP6_ECHO_REQUEST_REPLY::TimeStamp, and _ICMP6_ECHO_REQUEST_REPLY::Type.

Referenced by Ping6SendEchoRequest().

```
EFI_STATUS Ping6GetFrequency ( VOID )
```

Get and calculate the frequency in tick/ms. The result is saved in the globle variable mFrequency

Return values:

EFI_SUCCESS Calculated the frequency successfully.
Others Failed to calculate the frequency.

References mFrequency, and NTTIMERPERIOD.

Referenced by InitializePing6().

```
EFI_STATUS Ping6MatchEchoReply ( IN PING6_PRIVATE_DATA *          Private,
                                 IN ICMP6_ECHO_REQUEST_REPLY *    Packet
                               )
```

Match the request, and reply with SequenceNum/TimeStamp.

Parameters:

[in] *Private* The pointer to PING6_PRIVATE_DATA.
[in] *Packet* The pointer to ICMP6_ECHO_REQUEST_REPLY.

Return values:

EFI_SUCCESS	The match is successful.
EFI_NOT_FOUND	The reply can't be matched with any request.

References _PING6_ICMP6_TX_INFO::Link, Ping6DestroyTxInfo(), _PING6_ICMP6_TX_INFO::SequenceNum, and _PING6_ICMP6_TX_INFO::TimeStamp.

Referenced by Ping6OnEchoReplyReceived().

```
VOID EFIAPI Ping6OnEchoReplyReceived ( IN EFI_EVENT    Event,
                                       IN VOID *       Context
                                     )
```

receive reply, match and print reply infomation.

Parameters:

[in]	*Event*	A EFI_EVENT type event.
[in]	*Context*	The pointer to context.

References _PING6_PRIVATE_DATA::BufferSize, _ICMP6_ECHO_REQUEST_REPLY::Code, _PING6_PRIVATE_DATA::DstAddress, _PING6_PRIVATE_DATA::Ip6, mHiiHandle, mIp6DstString, Ping6CalculateTick(), Ping6MatchEchoReply(), ReadTime(), _PING6_PRIVATE_DATA::RttMax, _PING6_PRIVATE_DATA::RttMin, _PING6_PRIVATE_DATA::RttSum, _PING6_PRIVATE_DATA::RxCount, _PING6_PRIVATE_DATA::RxToken, _PING6_PRIVATE_DATA::SendNum, _ICMP6_ECHO_REQUEST_REPLY::SequenceNum, _PING6_PRIVATE_DATA::Status, _ICMP6_ECHO_REQUEST_REPLY::TimeStamp, and _ICMP6_ECHO_REQUEST_REPLY::Type.

Referenced by Ping6ReceiveEchoReply().

```
VOID EFIAPI Ping6OnEchoRequestSent ( IN EFI_EVENT    Event,
                                     IN VOID *       Context
                                   )
```

The original intention is to send a request. Currently, the application retransmits an icmp6 echo request packet per second in sendnumber times that is specified by the user. Because nothing can be done here, all things move to the timer rountine.

Parameters:

[in]	*Event*	A EFI_EVENT type event.
[in]	*Context*	The pointer to Context.

Referenced by Ping6GenerateToken().

```
VOID EFIAPI Ping6OnTimerRoutine ( IN EFI_EVENT    Event,
                                  IN VOID *       Context
                                )
```

Remove the timeout request from the list.

Parameters:

[in] *Event* A EFI_EVENT type event.
[in] *Context* The pointer to Context.

References _PING6_PRIVATE_DATA::Ip6, _PING6_ICMP6_TX_INFO::Link, mHiiHandle, PING6_DEFAULT_TIMEOUT, Ping6CalculateTick(), Ping6DestroyTxInfo(), Ping6SendEchoRequest(), ReadTime(), _PING6_PRIVATE_DATA::SendNum, _PING6_ICMP6_TX_INFO::SequenceNum, _PING6_PRIVATE_DATA::Status, _PING6_ICMP6_TX_INFO::TimeStamp, _PING6_ICMP6_TX_INFO::Token, _PING6_PRIVATE_DATA::TxCount, and _PING6_PRIVATE_DATA::TxList.

Referenced by Ping6().

EFI_STATUS Ping6ReceiveEchoReply (IN PING6_PRIVATE_DATA * *Private*)
Place a completion token into the receive packet queue to receive the echo reply.

Parameters:
[in] *Private* The pointer of PING6_PRIVATE_DATA.

Return values:
EFI_SUCCESS Put the token into the receive packet queue successfully.
others Put the token into the receive packet queue unsuccessfully.

References Ping6OnEchoReplyReceived().

Referenced by Ping6().

EFI_STATUS Ping6SendEchoRequest (IN PING6_PRIVATE_DATA * *Private*)
Transmit the EFI_IP6_COMPLETION_TOKEN.

Parameters:
[in] *Private* The pointer of PING6_PRIVATE_DATA.

Return values:
EFI_SUCCESS Transmitted successfully.
EFI_OUT_OF_RESOURCES No memory is available on the platform.
others Transmitted unsuccessfully.

References _PING6_ICMP6_TX_INFO::Link, Ping6DestroyTxInfo(), Ping6GenerateToken(), ReadTime(), _PING6_ICMP6_TX_INFO::SequenceNum, _PING6_ICMP6_TX_INFO::TimeStamp, and _PING6_ICMP6_TX_INFO::Token.

Referenced by Ping6(), and Ping6OnTimerRoutine().

Variable Documentation

UINT64 mFrequency = 0

Referenced by Ping6CalculateTick(), and Ping6GetFrequency().

EFI_HII_HANDLE mHiiHandle

CONST CHAR16* mIp6DstString

Referenced by InitializePing6(), Ping6(), and Ping6OnEchoReplyReceived().

CONST CHAR16* mIp6SrcString

Referenced by InitializePing6(), and Ping6CreateIp6Instance().

SHELL_PARAM_ITEM Ping6ParamList[]

Initial value:

```
 {
  {
    L"-l",
    TypeValue
  },
  {
    L"-n",
    TypeValue
  },
  {
    L"-s",
    TypeValue
  },
  {
    L"-?",
    TypeFlag
  },
  {
    NULL,
    TypeMax
  },
}
```

Referenced by InitializePing6().

NetworkPkg/Application/Ping6/Ping6.h File Reference

Data Structures

struct _ICMP6_ECHO_REQUEST_REPLY
struct _PING6_ICMP6_TX_INFO
struct _PING6_PRIVATE_DATA

Defines

"memItemRight" valign="bottom">PING6_DEFAULT_TIMEOUT 5000
#define PING6_MAX_SEND_NUMBER 10000
#define PING6_MAX_BUFFER_SIZE 32768
#define PING6_ONE_SECOND 10000000
#define NTTIMERPERIOD 358049

Typedefs

typedef struct _ICMP6_ECHO_REQUEST_REPLY ICMP6_ECHO_REQUEST_REPLY
typedef struct _PING6_ICMP6_TX_INFO PING6_ICMP6_TX_INFO
typedef struct _PING6_PRIVATE_DATA PING6_PRIVATE_DATA

Functions

UINT64 ReadTime (VOID)

Detailed Description

The interface function declaration of shell application Ping6 (Ping for v6 series).

Copyright (c) 2009 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Ping6GetFrequency().

#define PING6_DEFAULT_TIMEOUT 5000

Referenced by Ping6OnTimerRoutine().

#define PING6_MAX_BUFFER_SIZE 32768

Referenced by InitializePing6().

#define PING6_MAX_SEND_NUMBER 10000

Referenced by InitializePing6().

#define PING6_ONE_SECOND 10000000

Referenced by Ping6().

Typedef Documentation

typedef struct _ICMP6_ECHO_REQUEST_REPLY ICMP6_ECHO_REQUEST_REPLY

typedef struct _PING6_ICMP6_TX_INFO PING6_ICMP6_TX_INFO

typedef struct _PING6_PRIVATE_DATA PING6_PRIVATE_DATA

Function Documentation

UINT64 ReadTime (VOID)

Reads and returns the current value of register. In IA64, the register is the Interval Timer Vector (ITV). In X86(IA32/X64), the register is the Time Stamp Counter (TSC)

Returns:

The current value of the register.

NetworkPkg/Application/Ping6/Ia32/Tsc.c File Reference

Functions

UINT64 ReadTime ()

Detailed Description

The implement to read TSC in IA32 platform.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

UINT64 ReadTime ( )

Reads and returns the current value of the Time Stamp Counter (TSC).

Returns:

The current value of TSC.

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Application/Ping6/Ipf/Itc.c File Reference

Functions

UINT64 ReadTime ()

Detailed Description

The implement to read ITC in IA64 platform.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

UINT64 ReadTime ( )

Reads and returns the current value of the Interval Timer Counter Register (ITC).

Returns:

The current value of ITC.

Referenced by Ping6OnEchoReplyReceived(), Ping6OnTimerRoutine(), and Ping6SendEchoRequest().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

NetworkPkg/Application/Ping6/X64/Tsc.c File Reference

Functions

UINT64 ReadTime ()

Detailed Description

The implement to read TSC in X64 platform.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

UINT64 ReadTime ( )

Reads and returns the current value of Time Stamp Counter (TSC).

Returns:

The current value of TSC

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Application/VConfig/VConfig.c File Reference

Defines

"memItemRight" valign="bottom">INVALID_NIC_INDEX 0xffff

#define INVALID_VLAN_ID 0xffff

Functions

VOID	LocateNicHandleBuffer (OUT UINTN *NumberOfHandles, OUT EFI_HANDLE **HandleBuffer)
UINTN	NicNameToIndex (IN CHAR16 *Name)
EFI_HANDLE	NicNameToHandle (IN CHAR16 *Name)
EFI_VLAN_CONFIG_PROTOCOL *	OpenVlanConfigProtocol (IN EFI_HANDLE Handle)
VOID	CloseVlanConfigProtocol (IN EFI_HANDLE Handle)
VOID	ShowNicVlanInfo (IN EFI_HANDLE Handle, IN UINTN NicIndex)
VOID	DisplayVlan (IN CHAR16 *Name)
UINT16	StrToVlanId (IN CHAR16 *String)
VOID	AddVlan (IN CHAR16 *ParamStr)
VOID	DeleteVlan (IN CHAR16 *ParamStr)
EFI_STATUS EFIAPI	VlanConfigMain (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)

Variables

UINT8	VConfigStrings []
EFI_HANDLE	mImageHandle = NULL
EFI_HII_HANDLE	mHiiHandle = NULL
SHELL_PARAM_ITEM	mParamList []

Detailed Description

Shell application for VLAN configuration.

Copyright (c) 2009 - 2013, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by NicNameToIndex().

#define INVALID_VLAN_ID 0xffff

Referenced by StrToVlanId().

Function Documentation

VOID AddVlan (IN CHAR16 * *ParamStr*)

Add a VLAN device.

Parameters:

[in] *ParamStr* Parameter string from user input.

References CloseVlanConfigProtocol(), mHiiHandle, NicNameToHandle(), OpenVlanConfigProtocol(), and StrToVlanId().

Referenced by VlanConfigMain().

VOID CloseVlanConfigProtocol (IN EFI_HANDLE *Handle*)

Close VlanConfig protocol of a handle.

Parameters:

[in] *Handle* The handle to close the VlanConfig protocol.

References mImageHandle.

Referenced by AddVlan(), DeleteVlan(), and ShowNicVlanInfo().

VOID DeleteVlan (IN CHAR16 * *ParamStr*)

Remove a VLAN device.

Parameters:

[in] *ParamStr* Parameter string from user input.

References CloseVlanConfigProtocol(), mHiiHandle, NicNameToHandle(), OpenVlanConfigProtocol(), and StrToVlanId().

Referenced by VlanConfigMain().

VOID DisplayVlan (IN CHAR16 * *Name*)

Display the VLAN configuration of all, or a specified network interface.

Parameters:

[in] *Name* Name of the network interface. If NULL, the VLAN configuration of all network will be displayed.

References LocateNicHandleBuffer(), NicNameToHandle(), and ShowNicVlanInfo().

Referenced by VlanConfigMain().

VOID LocateNicHandleBuffer (OUT UINTN * *NumberOfHandles*,
OUT EFI_HANDLE ** *HandleBuffer*
)

Locate the network interface handle buffer.

Parameters:

[out] *NumberOfHandles* Pointer to the number of handles.

[out] *HandleBuffer* Pointer to the buffer to store the returned handles.

References mHiiHandle.

Referenced by DisplayVlan(), and NicNameToHandle().

EFI_HANDLE NicNameToHandle (IN CHAR16 * *Name*)

Find network interface device handle by its name.

Parameters:

[in] *Name* Name of the network interface.

Return values:

NULL Cannot find the network interface.

Returns:

others Handle of the network interface.

References LocateNicHandleBuffer(), mHiiHandle, and NicNameToIndex().

Referenced by AddVlan(), DeleteVlan(), and DisplayVlan().

UINTN NicNameToIndex (IN CHAR16 * *Name*)

Extract the decimal index from the network interface name.

Parameters:

[in] *Name* Name of the network interface.

Return values:

INVALID_NIC_INDEX Failed to extract the network interface index.

Returns:

others The network interface index.

References INVALID_NIC_INDEX.

Referenced by NicNameToHandle().

EFI_VLAN_CONFIG_PROTOCOL* OpenVlanConfigProtocol (IN EFI_HANDLE *Handle*)

Open VlanConfig protocol from a handle.

Parameters:

[in] *Handle* The handle to open the VlanConfig protocol.

Returns:

The VlanConfig protocol interface.

References mImageHandle.

Referenced by AddVlan(), DeleteVlan(), and ShowNicVlanInfo().

VOID ShowNicVlanInfo (IN EFI_HANDLE *Handle*,
IN UINTN *NicIndex*
)

Display VLAN configuration of a network interface.

Parameters:

[in] *Handle* Handle of the network interface.

[in] *NicIndex* Index of the network interface.

References CloseVlanConfigProtocol(), mHiiHandle, mImageHandle, and OpenVlanConfigProtocol().

Referenced by DisplayVlan().

UINT16 StrToVlanId (IN CHAR16 * *String*)

Convert a NULL-terminated unicode decimal VLAN ID string to VLAN ID.

Parameters:

[in] *String* Pointer to VLAN ID string from user input.

Return values:

Value translated from String, or INVALID_VLAN_ID is string is invalid.

References INVALID_VLAN_ID.

Referenced by AddVlan(), and DeleteVlan().

EFI_STATUS EFIAPI VlanConfigMain (IN EFI_HANDLE *ImageHandle*,
IN EFI_SYSTEM_TABLE * *SystemTable*
)

The actual entry point for the application.

Parameters:

[in] *ImageHandle* The firmware allocated handle for the EFI image.

[in] *SystemTable* A pointer to the EFI System Table.

Return values:

EFI_SUCCESS	The entry point executed successfully.
other	Some error occur when executing this entry point.

References AddVlan(), DeleteVlan(), DisplayVlan(), mHiiHandle, mImageHandle, mParamList, and VConfigStrings.

Variable Documentation

EFI_HII_HANDLE mHiiHandle = NULL

EFI_HANDLE mImageHandle = NULL

Referenced by CloseVlanConfigProtocol(), OpenVlanConfigProtocol(), ShowNicVlanInfo(), and VlanConfigMain().

SHELL_PARAM_ITEM mParamList[]

Initial value:

```
 {
  {
    L"-l",
    TypeValue
  },
  {
    L"-a",
    TypeMaxValue
  },
  {
    L"-d",
    TypeValue
  },
  {
    NULL,
    TypeMax
  }
}
```

Referenced by VlanConfigMain().

UINT8 VConfigStrings[]

Referenced by VlanConfigMain().

NetworkPkg/Dhcp6Dxe/ComponentName.c File Reference

Functions

EFI_STATUS EFIAPI Dhcp6ComponentNameGetDriverName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN CHAR8 *Language, OUT CHAR16 **DriverName)

EFI_STATUS EFIAPI Dhcp6ComponentNameGetControllerName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE ChildHandle, IN CHAR8 *Language, OUT CHAR16 **ControllerName)

EFI_STATUS UpdateName (IN EFI_DHCP6_PROTOCOL *Dhcp6)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gDhcp6ComponentName

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gDhcp6ComponentName2

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mDhcp6DriverNameTable []

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE * gDhcp6ControllerNameTable = NULL

CHAR16 * mDhcp6ControllerName []

Detailed Description

UEFI Component Name(2) protocol implementation for Dhcp6 driver.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI Dhcp6ComponentNameGetControllerName (	IN EFI_COMPONENT_NAME_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_HANDLE	*ChildHandle*,

```
  IN CHAR8 *                    Language,
  OUT CHAR16 **                 ControllerName
)
```

Retrieves a Unicode string that is the user-readable name of the controller that is being managed by a driver.

This function retrieves the user-readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
[in]	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that attempt to retrieve the name of the bus controller. It will not be NULL for a bus driver that attempts to retrieve the name of a child controller.
[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language, from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL, and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

Retrieves a Unicode string that is the user-readable name of the controller that is being managed by a driver.

This function retrieves the user-readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
[in]	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in the RFC 4646 or ISO 639-2 language code format.
[out]	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language, from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL, and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gDhcp6ComponentName, gDhcp6ControllerNameTable, and UpdateName().

```
EFI_STATUS EFIAPI
Dhcp6ComponentNameGetDriverName
  (
    IN EFI_COMPONENT_NAME_PROTOCOL *   This,
    IN CHAR8 *                         Language,
    OUT CHAR16 **                      DriverName
  )
```

Retrieves a Unicode string that is the user-readable name of the driver.

This function retrieves the user-readable name of a driver in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the driver name is returned in DriverName, and EFI_SUCCESS is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.

[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*DriverName*	A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by This in the language specified by Language.

Return values:

EFI_SUCCESS	The Unicode string for the Driver specified by This and the language specified by Language was returned in DriverName.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	DriverName is NULL.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gDhcp6ComponentName, and mDhcp6DriverNameTable.

EFI_STATUS UpdateName (IN EFI_DHCP6_PROTOCOL * *Dhcp6*)

Update the component name for the Dhcp6 child handle.

Parameters:

Dhcp6[in] A pointer to the EFI_DHCP6_PROTOCOL.

Return values:

EFI_SUCCESS	Update the ControllerNameTable of this instance successfully.
EFI_INVALID_PARAMETER	The input parameter is invalid.

References gDhcp6ComponentName, gDhcp6ComponentName2, gDhcp6ControllerNameTable, and mDhcp6ControllerName.

Referenced by Dhcp6ComponentNameGetControllerName(), Ip6ComponentNameGetControllerName(), IScsiComponentNameGetControllerName(), Mtftp6ComponentNameGetControllerName(), and Udp6ComponentNameGetControllerName().

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gDhcp6ComponentName

Initial value:

```
{
 Dhcp6ComponentNameGetDriverName,
 Dhcp6ComponentNameGetControllerName,
 "eng"
}
```

Referenced by Dhcp6ComponentNameGetControllerName(), Dhcp6ComponentNameGetDriverName(), Dhcp6DriverEntryPoint(), and UpdateName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gDhcp6ComponentName2

Initial value:

```
{
  (EFI_COMPONENT_NAME2_GET_DRIVER_NAME) Dhcp6ComponentNameGetDriverName,
  (EFI_COMPONENT_NAME2_GET_CONTROLLER_NAME) Dhcp6ComponentNameGetControllerName,
  "en"
}
```

Referenced by Dhcp6DriverEntryPoint(), and UpdateName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE* gDhcp6ControllerNameTable = NULL

Referenced by Dhcp6ComponentNameGetControllerName(), and UpdateName().

CHAR16* mDhcp6ControllerName[]

Initial value:

```
{
 L"DHCPv6 (State=0, Init)",
 L"DHCPv6 (State=1, Selecting)",
 L"DHCPv6 (State=2, Requesting)",
 L"DHCPv6 (State=3, Declining)",
 L"DHCPv6 (State=4, Confirming)",
 L"DHCPv6 (State=5, Releasing)",
 L"DHCPv6 (State=6, Bound)",
 L"DHCPv6 (State=7, Renewing)",
 L"DHCPv6 (State=8, Rebinding)"
}
```

Referenced by UpdateName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mDhcp6DriverNameTable[]

Initial value:

```
{
 {
   "eng;en",
   L"DHCP6 Protocol Driver"
 },
 {
   NULL,
   NULL
 }
}
```

Referenced by Dhcp6ComponentNameGetDriverName().

NetworkPkg/Dhcp6Dxe/Dhcp6Driver.c File Reference

Functions

EFI_STATUS EFIAPI Dhcp6ConfigureUdpIo (IN UDP_IO *UdpIo, IN VOID *Context)
VOID Dhcp6DestroyService (IN OUT DHCP6_SERVICE *Service)
EFI_STATUS Dhcp6CreateService (IN EFI_HANDLE Controller, IN EFI_HANDLE ImageHandle, OUT DHCP6_SERVICE **Service)
VOID Dhcp6DestroyInstance (IN OUT DHCP6_INSTANCE *Instance)
EFI_STATUS Dhcp6CreateInstance (IN DHCP6_SERVICE *Service, OUT DHCP6_INSTANCE **Instance)
EFI_STATUS EFIAPI Dhcp6DestroyChildEntry (IN LIST_ENTRY *Entry, IN VOID *Context)
EFI_STATUS EFIAPI Dhcp6DriverEntryPoint (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)
EFI_STATUS EFIAPI Dhcp6DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI Dhcp6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI Dhcp6DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)
EFI_STATUS EFIAPI Dhcp6ServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN OUT EFI_HANDLE *ChildHandle)
EFI_STATUS EFIAPI Dhcp6ServiceBindingDestroyChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE ChildHandle)

Variables

EFI_DRIVER_BINDING_PROTOCOL gDhcp6DriverBinding
EFI_SERVICE_BINDING_PROTOCOL gDhcp6ServiceBindingTemplate

Detailed Description

Driver Binding functions and Service Binding functions implementationfor for Dhcp6 Driver.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI Dhcp6ConfigureUdpIo ( IN UDP_IO *   UdpIo,
                                        IN VOID *     Context
                                      )
```

Configure the default Udp6Io to receive all the DHCP6 traffic on this network interface.

Parameters:

[in]	*UdpIo*	The pointer to Udp6Io to be configured.
[in]	*Context*	The pointer to the context.

Return values:

EFI_SUCCESS	The Udp6Io is successfully configured.
Others	Failed to configure the Udp6Io.

References DHCP6_PORT_CLIENT.

Referenced by Dhcp6CreateService().

```
EFI_STATUS Dhcp6CreateInstance ( IN DHCP6_SERVICE *        Service,
                                 OUT DHCP6_INSTANCE **     Instance
                               )
```

Create the Dhcp6 instance and initialize it.

Parameters:

[in]	*Service*	The pointer to the Dhcp6 service.
[out]	*Instance*	The pointer to the Dhcp6 instance.

Return values:

EFI_SUCCESS	The Dhcp6 instance is created.
EFI_OUT_OF_RESOURCES	Failed to allocate resources.

References _DHCP6_INSTANCE::Dhcp6, DHCP6_INSTANCE_SIGNATURE, Dhcp6OnTimerTick(), gDhcp6ProtocolTemplate, _DHCP6_INSTANCE::InfDestroy, _DHCP6_INSTANCE::InfList, _DHCP6_INSTANCE::MediaPresent, _DHCP6_INSTANCE::Service, _DHCP6_INSTANCE::Signature, _DHCP6_INSTANCE::Timer, _DHCP6_INSTANCE::TxList, and _DHCP6_INSTANCE::UdpSts.

Referenced by Dhcp6ServiceBindingCreateChild().

```
EFI_STATUS Dhcp6CreateService ( IN EFI_HANDLE              Controller,
                                IN EFI_HANDLE              ImageHandle,
                                OUT DHCP6_SERVICE **       Service
                              )
```

Create a new Dhcp6 service for the Nic controller.

Parameters:

[in]	*Controller*	The controller to be installed DHCP6 service binding protocol.
[in]	*ImageHandle*	The image handle of the Dhcp6 driver.
[out]	*Service*	The return pointer of the new Dhcp6 service.

Return values:

EFI_SUCCESS	The Dhcp6 service is created successfully.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_OUT_OF_RESOURCES	Failed to allocate resource.

References _DHCP6_SERVICE::Child, _DHCP6_SERVICE::ClientId, _DHCP6_SERVICE::Controller, DHCP6_SERVICE_SIGNATURE, Dhcp6ConfigureUdpIo(), Dhcp6GenerateClientId(), gDhcp6ServiceBindingTemplate, _DHCP6_SERVICE::Image, _DHCP6_SERVICE::Ip6Cfg, _DHCP6_SERVICE::ServiceBinding, _DHCP6_SERVICE::Signature, _DHCP6_SERVICE::Snp, _DHCP6_SERVICE::UdpIo, and _DHCP6_SERVICE::Xid.

Referenced by Dhcp6DriverBindingStart().

EFI_STATUS EFIAPI Dhcp6DestroyChildEntry (IN LIST_ENTRY * *Entry*,
IN VOID * *Context*
)

Callback function which provided by user to remove one node in NetDestroyLinkList process.

Parameters:

[in]	*Entry*	The entry to be removed.
[in]	*Context*	Pointer to the callback context corresponds to the Context in NetDestroyLinkList.

Return values:

EFI_SUCCESS	The entry has been removed successfully.
Others	Fail to remove the entry.

References DHCP6_INSTANCE_SIGNATURE, and _DHCP6_INSTANCE::Handle.

Referenced by Dhcp6DriverBindingStop().

VOID Dhcp6DestroyInstance (IN OUT DHCP6_INSTANCE * *Instance*)

Destroy the Dhcp6 instance and recycle the resources.

Parameters:

[in,out]	*Instance*	The pointer to the Dhcp6 instance.

References DHCP6_PACKET_ALL, Dhcp6CleanupConfigData(), and Dhcp6CleanupRetry().

Referenced by Dhcp6ServiceBindingCreateChild(), and Dhcp6ServiceBindingDestroyChild().

VOID Dhcp6DestroyService (IN OUT DHCP6_SERVICE * *Service*)

Destroy the Dhcp6 service. The Dhcp6 service may be partly initialized, or partly destroyed. If a resource is destroyed, it is marked as such in case the destroy failed and being called again later.

Parameters:

`[in,out]` *Service* The pointer to Dhcp6 service to be destroyed.

Referenced by Dhcp6DriverBindingStart(), and Dhcp6DriverBindingStop().

EFI_STATUS EFIAPI Dhcp6DriverBindingStart	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*
	)	

Start this driver on ControllerHandle. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Start() it must also follow these calling restrictions.

Parameters:

`[in]`	*This*	The pointer to the driver binding protocol.
`[in]`	*ControllerHandle*	The handle of device to be started.
`[in]`	*RemainingDevicePath*	Optional parameter use to pick a specific child device to be started.

Return values:

EFI_SUCCESS	This driver is installed to ControllerHandle.
EFI_ALREADY_STARTED	This driver is already running on ControllerHandle.
other	This driver does not support this device.

References Dhcp6CreateService(), Dhcp6DestroyService(), and _DHCP6_SERVICE::ServiceBinding.

EFI_STATUS EFIAPI Dhcp6DriverBindingStop	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN UINTN	*NumberOfChildren*,
	IN EFI_HANDLE *	*ChildHandleBuffer*
	)	

Stop this driver on ControllerHandle. This service is called by the EFI boot service DisconnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. DisconnectController() must follow these calling restrictions. If any other agent wishes to call Stop() it must also follow these calling restrictions.

Parameters:

`[in]`	*This*	Protocol instance pointer.
`[in]`	*ControllerHandle*	Handle of device to stop driver on
`[in]`	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If number of children is zero stop the entire bus driver.
`[in]`	*ChildHandleBuffer*	List of Child Handles to Stop.

Return values:

EFI_SUCCESS	This driver is removed ControllerHandle
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
other	This driver was not removed from this device

References _DHCP6_SERVICE::Child, DHCP6_SERVICE_FROM_THIS, Dhcp6DestroyChildEntry(), and Dhcp6DestroyService().

```
EFI_STATUS EFIAPI
Dhcp6DriverBindingSupported
  ( IN EFI_DRIVER_BINDING_PROTOCOL *   This,
    IN EFI_HANDLE                      ControllerHandle,
    IN EFI_DEVICE_PATH_PROTOCOL *      RemainingDevicePath
  )
```

Test to see if this driver supports ControllerHandle. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Supported() it must also follow these calling restrictions.

Parameters:

[in]	*This*	The pointer to the driver binding protocol.
[in]	*ControllerHandle*	The handle of device to be tested.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to be started.

Return values:

EFI_SUCCESS	This driver supports this device.
Others	This driver does not support this device.

```
EFI_STATUS EFIAPI Dhcp6DriverEntryPoint ( IN EFI_HANDLE        ImageHandle,
                                          IN EFI_SYSTEM_TABLE * SystemTable
                                        )
```

Entry point of the DHCP6 driver to install various protocols.

Parameters:

[in]	*ImageHandle*	The handle of the UEFI image file.
[in]	*SystemTable*	The pointer to the EFI System Table.

Return values:

EFI_SUCCESS	The operation completed successfully.
Others	Unexpected error occurs.

References gDhcp6ComponentName, gDhcp6ComponentName2, and gDhcp6DriverBinding.

```
EFI_STATUS EFIAPI
Dhcp6ServiceBindingCreateChild
  ( IN EFI_SERVICE_BINDING_PROTOCOL *   This,
    IN OUT EFI_HANDLE *                 ChildHandle
  )
```

Creates a child handle and installs a protocol.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

[in]	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
[in,out]	*ChildHandle*	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.
EFI_INVALID_PARAMETER	ChildHandle is NULL.
other	The child handle was not created.

References _DHCP6_SERVICE::Child, _DHCP6_INSTANCE::Dhcp6, DHCP6_SERVICE_FROM_THIS, Dhcp6CreateInstance(), Dhcp6DestroyInstance(), gDhcp6DriverBinding, _DHCP6_INSTANCE::Handle, _DHCP6_INSTANCE::Link, _DHCP6_SERVICE::NumOfChild, _DHCP6_INSTANCE::Timer, and _DHCP6_SERVICE::UdpIo.

EFI_STATUS EFIAPI Dhcp6ServiceBindingDestroyChild (IN EFI_SERVICE_BINDING_PROTOCOL * *This*, IN EFI_HANDLE *ChildHandle*)

Destroys a child handle with a protocol installed on it.

The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

[in]	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
[in]	*ChildHandle*	Handle of the child to destroy

Return values:

EFI_SUCCES	The protocol was removed from ChildHandle.
EFI_UNSUPPORTED	ChildHandle does not support the protocol that is being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
EFI_ACCESS_DENIED	The protocol could not be removed from the ChildHandle because its services are being used.
other	The child handle was not destroyed

References DHCP6_INSTANCE_FROM_THIS, DHCP6_SERVICE_FROM_THIS, Dhcp6DestroyInstance(), gDhcp6DriverBinding, _DHCP6_INSTANCE::InDestroy, _DHCP6_INSTANCE::Link, _DHCP6_SERVICE::NumOfChild, _DHCP6_INSTANCE::Service, and _DHCP6_SERVICE::UdpIo.

Variable Documentation

EFI_DRIVER_BINDING_PROTOCOL gDhcp6DriverBinding

Initial value:

```
{
 Dhcp6DriverBindingSupported,
 Dhcp6DriverBindingStart,
 Dhcp6DriverBindingStop,
 0xa,
```

```
  NULL,
  NULL
}
```

Referenced by Dhcp6DriverEntryPoint(), Dhcp6ServiceBindingCreateChild(), and Dhcp6ServiceBindingDestroyChild().

EFI_SERVICE_BINDING_PROTOCOL gDhcp6ServiceBindingTemplate

Initial value:

```
 {
  Dhcp6ServiceBindingCreateChild,
  Dhcp6ServiceBindingDestroyChild
}
```

Referenced by Dhcp6CreateService().

NetworkPkg/Dhcp6Dxe/Dhcp6Driver.h File Reference

Functions

EFI_STATUS EFIAPI Dhcp6DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI Dhcp6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI Dhcp6DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)
EFI_STATUS EFIAPI Dhcp6ServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN OUT EFI_HANDLE *ChildHandle)
EFI_STATUS EFIAPI Dhcp6ServiceBindingDestroyChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE ChildHandle)

Variables

EFI_COMPONENT_NAME_PROTOCOL gDhcp6ComponentName
EFI_COMPONENT_NAME2_PROTOCOL gDhcp6ComponentName2
EFI_UNICODE_STRING_TABLE * gDhcp6ControllerNameTable

Detailed Description

Driver Binding functions and Service Binding functions declaration for Dhcp6 Driver.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI Dhcp6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL * *This*,

```
    IN EFI_HANDLE                    ControllerHandle,
    IN EFI_DEVICE_PATH_PROTOCOL *    RemainingDevicePath
)
```

Start this driver on ControllerHandle. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Start(), it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to bind driver to.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	This driver is added to ControllerHandle.
EFI_ALREADY_STARTED	This driver is already running on ControllerHandle.
other	This driver does not support this device.

Start this driver on ControllerHandle. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Start() it must also follow these calling restrictions.

Parameters:

[in]	*This*	The pointer to the driver binding protocol.
[in]	*ControllerHandle*	The handle of device to be started.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to be started.

Return values:

EFI_SUCCESS	This driver is installed to ControllerHandle.
EFI_ALREADY_STARTED	This driver is already running on ControllerHandle.
other	This driver does not support this device.

References Dhcp6CreateService(), Dhcp6DestroyService(), and _DHCP6_SERVICE::ServiceBinding.

```
EFI_STATUS EFIAPI
Dhcp6DriverBindingStop (
    IN EFI_DRIVER_BINDING_PROTOCOL *    This,
    IN EFI_HANDLE                       ControllerHandle,
    IN UINTN                            NumberOfChildren,
    IN EFI_HANDLE *                     ChildHandleBuffer
)
```

Stop this driver on ControllerHandle. This service is called by the EFI boot service DisconnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. DisconnectController() must follow these calling restrictions. If any other agent wishes to call Stop() it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to stop driver on.
[in]	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If the number of children is zero, stop the entire bus driver.

[in] *ChildHandleBuffer* List of Child Handles to Stop.

Return values:

EFI_SUCCESS This driver is removed ControllerHandle.
other This driver was not removed from this device.

Stop this driver on ControllerHandle. This service is called by the EFI boot service DisconnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. DisconnectController() must follow these calling restrictions. If any other agent wishes to call Stop() it must also follow these calling restrictions.

Parameters:

[in] *This* Protocol instance pointer.
[in] *ControllerHandle* Handle of device to stop driver on
[in] *NumberOfChildren* Number of Handles in ChildHandleBuffer. If number of children is zero stop the entire bus driver.
[in] *ChildHandleBuffer* List of Child Handles to Stop.

Return values:

EFI_SUCCESS This driver is removed ControllerHandle
EFI_DEVICE_ERROR An unexpected system or network error occurred.
other This driver was not removed from this device

References _DHCP6_SERVICE::Child, DHCP6_SERVICE_FROM_THIS, Dhcp6DestroyChildEntry(), and Dhcp6DestroyService().

```
EFI_STATUS EFIAPI
Dhcp6DriverBindingSupported
  ( IN EFI_DRIVER_BINDING_PROTOCOL *   This,
    IN EFI_HANDLE                      ControllerHandle,
    IN EFI_DEVICE_PATH_PROTOCOL *      RemainingDevicePath
  )
```

Test to see if this driver supports ControllerHandle. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Supported(), it must also follow these calling restrictions.

Parameters:

[in] *This* Protocol instance pointer.
[in] *ControllerHandle* Handle of device to test.
[in] *RemainingDevicePath* Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS This driver supports this device.
EFI_ALREADY_STARTED This driver is already running on this device.
other This driver does not support this device.

Test to see if this driver supports ControllerHandle. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Supported() it must also follow these calling restrictions.

Parameters:

[in]	*This*	The pointer to the driver binding protocol.
[in]	*ControllerHandle*	The handle of device to be tested.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to be started.

Return values:

EFI_SUCCESS	This driver supports this device.
Others	This driver does not support this device.

EFI_STATUS EFIAPI Dhcp6ServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL * *This*, IN OUT EFI_HANDLE * *ChildHandle*)

Creates a child handle and installs a protocol.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

This	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
ChildHandle	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.
EFI_INVALID_PARAMETER	ChildHandle is NULL.
EFI_OUT_OF_RESOURCES	There are not enough resources availabe to create the child.
other	The child handle was not created.

Creates a child handle and installs a protocol.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

[in]	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
[in,out]	*ChildHandle*	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.
EFI_INVALID_PARAMETER	ChildHandle is NULL.
other	The child handle was not created.

References _DHCP6_SERVICE::Child, _DHCP6_INSTANCE::Dhcp6, DHCP6_SERVICE_FROM_THIS, Dhcp6CreateInstance(), Dhcp6DestroyInstance(), gDhcp6DriverBinding, _DHCP6_INSTANCE::Handle, _DHCP6_INSTANCE::Link, _DHCP6_SERVICE::NumOfChild, _DHCP6_INSTANCE::Timer, and _DHCP6_SERVICE::UdpIo.

EFI_STATUS EFIAPI Dhcp6ServiceBindingDestroyChild	(IN EFI_SERVICE_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ChildHandle*
	)	

Destroys a child handle with a protocol installed on it.

The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

This	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
ChildHandle	Handle of the child to destroy.

Return values:

EFI_SUCCES	The protocol was removed from ChildHandle.
EFI_UNSUPPORTED	ChildHandle does not support the protocol that is being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
EFI_ACCESS_DENIED	The protocol could not be removed from the ChildHandle because its services are being used.
other	The child handle was not destroyed

Destroys a child handle with a protocol installed on it.

The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

`[in]`	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
`[in]`	*ChildHandle*	Handle of the child to destroy

Return values:

EFI_SUCCES	The protocol was removed from ChildHandle.
EFI_UNSUPPORTED	ChildHandle does not support the protocol that is being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
EFI_ACCESS_DENIED	The protocol could not be removed from the ChildHandle because its services are being used.
other	The child handle was not destroyed

References DHCP6_INSTANCE_FROM_THIS, DHCP6_SERVICE_FROM_THIS, Dhcp6DestroyInstance(), gDhcp6DriverBinding, _DHCP6_INSTANCE::InDestroy, _DHCP6_INSTANCE::Link, _DHCP6_SERVICE::NumOfChild, _DHCP6_INSTANCE::Service, and _DHCP6_SERVICE::UdpIo.

Variable Documentation

EFI_COMPONENT_NAME_PROTOCOL gDhcp6ComponentName

Referenced by Dhcp6ComponentNameGetControllerName(), Dhcp6ComponentNameGetDriverName(), Dhcp6DriverEntryPoint(), and UpdateName().

EFI_COMPONENT_NAME2_PROTOCOL gDhcp6ComponentName2

Referenced by Dhcp6DriverEntryPoint(), and UpdateName().

EFI_UNICODE_STRING_TABLE* gDhcp6ControllerNameTable

Referenced by Dhcp6ComponentNameGetControllerName(), and UpdateName().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Dhcp6Dxe/Dhcp6Impl.c File Reference

Functions

EFI_STATUS EFIAPI EfiDhcp6Start (IN EFI_DHCP6_PROTOCOL *This)
EFI_STATUS EFIAPI EfiDhcp6Stop (IN EFI_DHCP6_PROTOCOL *This)
EFI_STATUS EFIAPI EfiDhcp6GetModeData (IN EFI_DHCP6_PROTOCOL *This, OUT EFI_DHCP6_MODE_DATA *Dhcp6ModeData, OUT EFI_DHCP6_CONFIG_DATA *Dhcp6ConfigData)
EFI_STATUS EFIAPI EfiDhcp6Configure (IN EFI_DHCP6_PROTOCOL *This, IN EFI_DHCP6_CONFIG_DATA *Dhcp6CfgData)
EFI_STATUS EFIAPI EfiDhcp6InfoRequest (IN EFI_DHCP6_PROTOCOL *This, IN BOOLEAN SendClientId, IN EFI_DHCP6_PACKET_OPTION *OptionRequest, IN UINT32 OptionCount, IN EFI_DHCP6_PACKET_OPTION *OptionList[], IN EFI_DHCP6_RETRANSMISSION *Retransmission, IN EFI_EVENT TimeoutEvent, IN EFI_DHCP6_INFO_CALLBACK ReplyCallback, IN VOID *CallbackContext)
EFI_STATUS EFIAPI EfiDhcp6RenewRebind (IN EFI_DHCP6_PROTOCOL *This, IN BOOLEAN RebindRequest)
EFI_STATUS EFIAPI EfiDhcp6Decline (IN EFI_DHCP6_PROTOCOL *This, IN UINT32 AddressCount, IN EFI_IPv6_ADDRESS *Addresses)
EFI_STATUS EFIAPI EfiDhcp6Release (IN EFI_DHCP6_PROTOCOL *This, IN UINT32 AddressCount, IN EFI_IPv6_ADDRESS *Addresses)
EFI_STATUS EFIAPI EfiDhcp6Parse (IN EFI_DHCP6_PROTOCOL *This, IN EFI_DHCP6_PACKET *Packet, IN OUT UINT32 *OptionCount, OUT EFI_DHCP6_PACKET_OPTION *PacketOptionList[])

Variables

EFI_IPv6_ADDRESS mAllDhcpRelayAndServersAddress = {{0xFF, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2}}
EFI_IPv6_ADDRESS mAllDhcpServersAddress = {{0xFF, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 3}}
EFI_DHCP6_PROTOCOL gDhcp6ProtocolTemplate

Detailed Description

This EFI_DHCP6_PROTOCOL interface implementation.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI EfiDhcp6Configure ( IN EFI_DHCP6_PROTOCOL *        This,
                                      IN EFI_DHCP6_CONFIG_DATA *     Dhcp6CfgData
                                    )
```

Initializes, changes, or resets the operational settings for the Dhcp6 instance.

The Configure() function is used to initialize or clean up the configuration data of the Dhcp6 instance:

- When Dhcp6CfgData is not NULL and Configure() is called successfully, the configuration data will be initialized in the Dhcp6 instance, and the state of the configured IA will be transferred into Dhcp6Init.
- When Dhcp6CfgData is NULL and Configure() is called successfully, the configuration data will be cleaned up and no IA will be associated with the Dhcp6 instance. To update the configuration data for an Dhcp6 instance, the original data must be cleaned up before setting the new configuration data.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol
[in]	*Dhcp6CfgData*	The pointer to the EFI_DHCP6_CONFIG_DATA.

Return values:

EFI_SUCCESS	The Dhcp6 is configured successfully with the Dhcp6Init state, or cleaned up the original configuration setting.
EFI_ACCESS_DENIED	The Dhcp6 instance was already configured. The Dhcp6 instance has already started the DHCPv6 S.A.R.R when Dhcp6CfgData is NULL.
EFI_INVALID_PARAMETER	Some of the parameter is invalid.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References _DHCP6_SERVICE::Child, _DHCP6_INSTANCE::Config, DHCP6_INSTANCE_FROM_THIS, Dhcp6CleanupConfigData(), Dhcp6CopyConfigData(), Dhcp6OptClientId, Dhcp6OptIana, Dhcp6OptIata, Dhcp6OptRapidCommit, Dhcp6OptReconfigureAccept, _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, and _DHCP6_INSTANCE::Service.

```
EFI_STATUS EFIAPI EfiDhcp6Decline ( IN EFI_DHCP6_PROTOCOL *   This,
                                    IN UINT32                 AddressCount,
                                    IN EFI_IPv6_ADDRESS *     Addresses
                                  )
```

Inform that one or more addresses assigned by a server are already in use by another node.

The Decline() function is used to manually decline the assignment of IPv6 addresses, which have been already used by another node. If all IPv6 addresses of the configured IA are declined through this function, the state of the IA will switch through Dhcp6Declining to Dhcp6Init. Otherwise, the state of the IA will restore to Dhcp6Bound after the declining process. The Decline() can only be called when the IA is in Dhcp6Bound state. If the EFI_DHCP6_CONFIG_DATA.IaInfoEvent is NULL, this function is a blocking operation. It will return after the declining process finishes, or aborted by user.

Parameters:

[in]	*This*	The pointer to EFI_DHCP6_PROTOCOL.
[in]	*AddressCount*	The number of declining addresses.

	[in] *Addresses*	The pointer to the buffer stored the declining addresses.

Return values:

EFI_SUCCESS	The DHCPv6 decline exchange process completed when EFI_DHCP6_CONFIG_DATA.IaInfoEvent was NULL. The Dhcp6 instance sent Decline packet when EFI_DHCP6_CONFIG_DATA.IaInfoEvent was not NULL.
EFI_ACCESS_DENIED	The Dhcp6 instance hasn't been configured, or the state of the configured IA is not in Dhcp6Bound.
EFI_ABORTED	The DHCPv6 decline exchange process aborted by user.
EFI_NOT_FOUND	Any specified IPv6 address is not correlated with the configured IA for this instance.
EFI_INVALID_PARAMETER	Some parameter is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References _DHCP6_INSTANCE::Config, DHCP6_INSTANCE_FROM_THIS, Dhcp6CheckAddress(), Dhcp6DepriveAddress(), Dhcp6ReceivePacket(), Dhcp6SendDeclineMsg(), _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, _DHCP6_INSTANCE::Service, _DHCP6_SERVICE::UdpIo, and _DHCP6_INSTANCE::UdpSts.

```
EFI_STATUS EFIAPI
EfiDhcp6GetModeData                    ( IN EFI_DHCP6_PROTOCOL *          This,
                                         OUT EFI_DHCP6_MODE_DATA *        Dhcp6ModeData,
                                         OUT EFI_DHCP6_CONFIG_DATA *      Dhcp6ConfigData
                                       )
```

Returns the current operating mode data for the Dhcp6 instance.

The GetModeData() function returns the current operating mode and cached data packet for the Dhcp6 instance.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol.
[out]	*Dhcp6ModeData*	The pointer to the Dhcp6 mode data.
[out]	*Dhcp6ConfigData*	The pointer to the Dhcp6 configure data.

Return values:

EFI_SUCCESS	The mode data was returned.
EFI_INVALID_PARAMETER	This is NULL.
EFI_ACCESS_DENIED	The EFI DHCPv6 Protocol instance was not configured.

References _DHCP6_SERVICE::ClientId, _DHCP6_INSTANCE::Config, DHCP6_INSTANCE_FROM_THIS, Dhcp6CleanupConfigData(), Dhcp6CleanupModeData(), Dhcp6CopyConfigData(), _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, and _DHCP6_INSTANCE::Service.

```
EFI_STATUS EFIAPI EfiDhcp6InfoRequest  ( IN EFI_DHCP6_PROTOCOL *          This,
                                         IN BOOLEAN                       SendClientId,
                                         IN EFI_DHCP6_PACKET_OPTION *     OptionRequest,
                                         IN UINT32                        OptionCount,
```

IN EFI_DHCP6_PACKET_OPTION *	*OptionList*[],
IN EFI_DHCP6_RETRANSMISSION *	*Retransmission*,
IN EFI_EVENT	*TimeoutEvent*,
IN EFI_DHCP6_INFO_CALLBACK	*ReplyCallback*,
IN VOID *	*CallbackContext*

)

Request configuration information without the assignment of any Ia addresses of the client.

The InfoRequest() function is used to request configuration information without the assignment of any IPv6 address of the client. The client sends out an Information Request packet to obtain the required configuration information, and DHCPv6 server responds with a Reply packet containing the information for the client. The received Reply packet will be passed to the user by ReplyCallback function. If the user returns EFI_NOT_READY from ReplyCallback, the Dhcp6 instance will continue to receive other Reply packets unless timeout according to the Retransmission parameter. Otherwise, the Information Request exchange process will be finished successfully if user returns EFI_SUCCESS from ReplyCallback.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol.
[in]	*SendClientId*	If TRUE, the DHCPv6 protocol instance will build Client Identifier option and include it into Information Request packet. Otherwise, Client Identifier option will not be included.
[in]	*OptionRequest*	The pointer to the buffer of option request options.
[in]	*OptionCount*	The option number in the OptionList.
[in]	*OptionList*	The list of appended options.
[in]	*Retransmission*	The pointer to the retransmission of the message.
[in]	*TimeoutEvent*	The event of timeout.
[in]	*ReplyCallback*	The callback function when the reply was received.
[in]	*CallbackContext*	The pointer to the parameter passed to the callback.

Return values:

EFI_SUCCESS	The DHCPv6 information request exchange process completed when TimeoutEvent is NULL. Information Request packet has been sent to DHCPv6 server when TimeoutEvent is not NULL.
EFI_NO_RESPONSE	The DHCPv6 information request exchange process failed because of no response, or not all requested-options are responded by DHCPv6 servers when Timeout happened.
EFI_ABORTED	The DHCPv6 information request exchange process was aborted by user.
EFI_INVALID_PARAMETER	Some parameter is NULL.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References DHCP6_INSTANCE_FROM_THIS, Dhcp6GetMappingTimeOut(), Dhcp6OptClientId, Dhcp6OptRequestOption, Dhcp6StartInfoRequest(), _DHCP6_SERVICE::Ip6Cfg, _DHCP6_INSTANCE::Service, _DHCP6_SERVICE::UdpIo, and _DHCP6_INSTANCE::UdpSts.

EFI_STATUS EFIAPI EfiDhcp6Parse (IN EFI_DHCP6_PROTOCOL * *This*,

```
                                    IN EFI_DHCP6_PACKET *                Packet,
                                    IN OUT UINT32 *                      OptionCount,
                                    OUT EFI_DHCP6_PACKET_OPTION *        PacketOptionList[]
                                  )
```

Parse the option data in the Dhcp6 packet.

The Parse() function is used to retrieve the option list in the DHCPv6 packet.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol.
[in]	*Packet*	The pointer to the Dhcp6 packet.
[in,out]	*OptionCount*	The number of option in the packet.
[out]	*PacketOptionList*	The array of pointers to each option in the packet.

Return values:

EFI_SUCCESS	The packet was successfully parsed.
EFI_INVALID_PARAMETER	Some parameter is NULL.
EFI_BUFFER_TOO_SMALL	*OptionCount is smaller than the number of options that were found in the Packet.

```
EFI_STATUS EFIAPI EfiDhcp6Release  ( IN EFI_DHCP6_PROTOCOL *   This,
                                     IN UINT32                 AddressCount,
                                     IN EFI_IPv6_ADDRESS *     Addresses
                                   )
```

Release one or more addresses associated with the configured Ia for current instance.

The Release() function is used to manually release one or more IPv6 addresses. If AddressCount is zero, it will release all IPv6 addresses of the configured IA. If all IPv6 addresses of the IA are released through this function, the state of the IA will switch through Dhcp6Releasing to Dhcp6Init, otherwise, the state of the IA will restore to Dhcp6Bound after the releasing process. The Release() can only be called when the IA is in Dhcp6Bound state. If the EFI_DHCP6_CONFIG_DATA.IaInfoEvent is NULL, the function is a blocking operation. It will return after the releasing process finishes, or is aborted by user.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol.
[in]	*AddressCount*	The number of releasing addresses.
[in]	*Addresses*	The pointer to the buffer stored the releasing addresses.

Return values:

EFI_SUCCESS	The DHCPv6 release exchange process completed when EFI_DHCP6_CONFIG_DATA.IaInfoEvent was NULL. The Dhcp6 instance was sent Release packet when EFI_DHCP6_CONFIG_DATA.IaInfoEvent was not NULL.
EFI_ACCESS_DENIED	The Dhcp6 instance hasn't been configured, or the state of the configured IA is not in Dhcp6Bound.
EFI_ABORTED	The DHCPv6 release exchange process aborted by user.
EFI_NOT_FOUND	Any specified IPv6 address is not correlated with the configured IA for this instance.
EFI_INVALID_PARAMETER	Some parameter is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References _DHCP6_INSTANCE::Config, DHCP6_INSTANCE_FROM_THIS, Dhcp6CheckAddress(), Dhcp6DepriveAddress(), Dhcp6ReceivePacket(), Dhcp6SendReleaseMsg(), _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, _DHCP6_INSTANCE::Service, _DHCP6_SERVICE::UdpIo, and _DHCP6_INSTANCE::UdpSts.

```
EFI_STATUS EFIAPI EfiDhcp6RenewRebind ( IN EFI_DHCP6_PROTOCOL *  This,
                                        IN BOOLEAN               RebindRequest
                                      )
```

Manually extend the valid and preferred lifetimes for the IPv6 addresses of the configured IA and update other configuration parameters by sending a Renew or Rebind packet.

The RenewRebind() function is used to manually extend the valid and preferred lifetimes for the IPv6 addresses of the configured IA, and update other configuration parameters by sending Renew or Rebind packet.

- When RebindRequest is FALSE and the state of the configured IA is Dhcp6Bound, it sends Renew packet to the previously DHCPv6 server and transfer the state of the configured IA to Dhcp6Renewing. If valid Reply packet received, the state transfers to Dhcp6Bound and the valid and preferred timer restarts. If fails, the state transfers to Dhcp6Bound, but the timer continues.
- When RebindRequest is TRUE and the state of the configured IA is Dhcp6Bound, it will send a Rebind packet. If valid Reply packet is received, the state transfers to Dhcp6Bound and the valid and preferred timer restarts. If it fails, the state transfers to Dhcp6Init, and the IA can't be used.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol.
[in]	*RebindRequest*	If TRUE, Rebind packet will be sent and enter Dhcp6Rebinding state. Otherwise, Renew packet will be sent and enter Dhcp6Renewing state.

Return values:

EFI_SUCCESS	The DHCPv6 renew/rebind exchange process has completed and at least one IPv6 address of the configured IA has been bound again when EFI_DHCP6_CONFIG_DATA.IaInfoEvent is NULL. The EFI DHCPv6 Protocol instance has sent Renew or Rebind packet when EFI_DHCP6_CONFIG_DATA.IaInfoEvent is not NULL.
EFI_ACCESS_DENIED	The Dhcp6 instance hasn't been configured, or the state of the configured IA is not in Dhcp6Bound.
EFI_ALREADY_STARTED	The state of the configured IA has already entered Dhcp6Renewing when RebindRequest is FALSE. The state of the configured IA has already entered Dhcp6Rebinding when RebindRequest is TRUE.
EFI_ABORTED	The DHCPv6 renew/rebind exchange process aborted by the user.
EFI_NO_RESPONSE	The DHCPv6 renew/rebind exchange process failed because of no response.
EFI_NO_MAPPING	No IPv6 address has been bound to the configured IA after the DHCPv6 renew/rebind exchange process.
EFI_INVALID_PARAMETER	Some parameter is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References _DHCP6_INSTANCE::Config, DHCP6_INSTANCE_FROM_THIS, Dhcp6ReceivePacket(), Dhcp6SendRenewRebindMsg(), _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, _DHCP6_INSTANCE::Service, _DHCP6_SERVICE::UdpIo, and _DHCP6_INSTANCE::UdpSts.

EFI_STATUS EFIAPI EfiDhcp6Start (IN EFI_DHCP6_PROTOCOL * *This*)
Starts the DHCPv6 standard S.A.R.R. process.

The Start() function starts the DHCPv6 standard process. This function can be called only when the state of Dhcp6 instance is in the Dhcp6Init state. If the DHCP process completes successfully, the state of the Dhcp6 instance will be transferred through Dhcp6Selecting and Dhcp6Requesting to the Dhcp6Bound state. Refer to rfc-3315 for precise state transitions during this process. At the time when each event occurs in this process, the callback function that was set by EFI_DHCP6_PROTOCOL.Configure() will be called, and the user can take this opportunity to control the process.

Parameters:

`[in]` *This* The pointer to Dhcp6 protocol.

Return values:

EFI_SUCCESS	The DHCPv6 standard process has started, or it has completed when CompletionEvent is NULL.
EFI_ACCESS_DENIED	The EFI DHCPv6 Child instance hasn't been configured.
EFI_INVALID_PARAMETER	This is NULL.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_TIMEOUT	The DHCPv6 configuration process failed because no response was received from the server within the specified timeout value.
EFI_ABORTED	The user aborted the DHCPv6 process.
EFI_ALREADY_STARTED	Some other Dhcp6 instance already started the DHCPv6 standard process.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_NO_MEDIA	There was a media error.

References _DHCP6_INSTANCE::Config, DHCP6_INSTANCE_FROM_THIS, Dhcp6ReceivePacket(), Dhcp6SendSolicitMsg(), _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, _DHCP6_INSTANCE::Service, _DHCP6_SERVICE::UdpIo, and _DHCP6_INSTANCE::UdpSts.

EFI_STATUS EFIAPI EfiDhcp6Stop (IN EFI_DHCP6_PROTOCOL * *This*)
Stops the DHCPv6 standard S.A.R.R. process.

The Stop() function is used to stop the DHCPv6 standard process. After this function is called successfully, the state of Dhcp6 instance is transferred into Dhcp6Init. EFI_DHCP6_PROTOCOL.Configure() needs to be called before DHCPv6 standard process can be started again. This function can be called when the Dhcp6 instance is in any state.

Parameters:

`[in]` *This* The pointer to the Dhcp6 protocol.

Return values:

EFI_SUCCESS	The Dhcp6 instance is now in the Dhcp6Init state.
EFI_INVALID_PARAMETER	This is NULL.

References _DHCP6_INSTANCE::Config, DHCP6_INSTANCE_FROM_THIS, Dhcp6CleanupSession(), Dhcp6SendReleaseMsg(), _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, _DHCP6_INSTANCE::Service, _DHCP6_SERVICE::UdpIo, and _DHCP6_INSTANCE::UdpSts.

Variable Documentation

EFI_DHCP6_PROTOCOL gDhcp6ProtocolTemplate

Initial value:

```
{
 EfiDhcp6GetModeData,
 EfiDhcp6Configure,
 EfiDhcp6Start,
 EfiDhcp6InfoRequest,
 EfiDhcp6RenewRebind,
 EfiDhcp6Decline,
 EfiDhcp6Release,
 EfiDhcp6Stop,
 EfiDhcp6Parse
}
```

Referenced by Dhcp6CreateInstance().

EFI_IPv6_ADDRESS mAllDhcpRelayAndServersAddress = {{0xFF, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2}}

Referenced by Dhcp6TransmitPacket(), and PxeBcRetryDhcp6Binl().

EFI_IPv6_ADDRESS mAllDhcpServersAddress = {{0xFF, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 3}}

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Dhcp6Dxe/Dhcp6Impl.h File Reference

Data Structures

struct _DHCP6_
struct _DHCP6_
struct _DHCP6_
struct _DHCP6_
struct _DHCP6_

Defines

"memItemRight" valign="bottom">DHCP6_SERVICE_SIGNATURE SIGNATURE_32 ('D', 'H', '6', 'S')
#define DHCP6_I
('D', 'H', '6
#define DHCP6_S
#define DHCP6_S
#define DHCP6_S
#define DHCP6_S
#define DHCP6_S
#define DHCP6_R
#define DHCP6_R
#define DHCP6_R
#define DHCP6_R
#define DHCP6_C
#define DHCP6_C
#define DHCP6_C
#define DHCP6_C
#define DHCP6_C
#define DHCP6_R
#define DHCP6_R
#define DHCP6_R
#define DHCP6_R
#define DHCP6_R
#define DHCP6_R
#define DHCP6_R
#define DHCP6_R
#define DHCP6_I
#define DHCP6_I
#define DHCP6_I
#define DHCP6_I
#define DHCP6_I
#define DHCP6_R
#define DHCP6_R
#define DHCP6_R
#define DHCP6_R
#define DHCP6_D
#define DHCP6_D
#define DHCP6_D
#define DHCP6_D
#define DHCP6_P
#define DHCP6_P

#define DHCP6_PACKET
#define DHCP6_BASE_P
#define DHCP6_PORT_C
#define DHCP6_PORT_S
#define DHCP6_INSTAN
((Instance), DHCP
Dhcp6,
DHCP6_INSTAN
#define DHCP6_SERVIC
((Service), DHCP6
ServiceBinding,
DHCP6_SERVIC

Typedefs

typedef struct _DHCP6_IA_CB DHCP6_IA_CB
typedef struct _DHCP6_INF_CB DHCP6_INF_CB
typedef struct _DHCP6_TX_CB DHCP6_TX_CB
typedef struct _DHCP6_SERVICE DHCP6_SERVIC
typedef struct _DHCP6_INSTANCE DHCP6_INSTAN

Enumerations

enum DHCP6_MSG_TY
Dhcp6MsgSolicit
Dhcp6MsgAdverti
Dhcp6MsgReques
Dhcp6MsgConfirm
Dhcp6MsgRenew
Dhcp6MsgRebind
Dhcp6MsgReply =
Dhcp6MsgRelease
Dhcp6MsgDeclin
Dhcp6MsgReconf
Dhcp6MsgInfoRe
}
enum DHCP6_OPT_CO
Dhcp6OptClientI
Dhcp6OptServerI
= 3, Dhcp6OptIata
Dhcp6OptIaAddr
Dhcp6OptRequest
Dhcp6OptPreferen
Dhcp6OptElapsed
Dhcp6OptReplay
Dhcp6OptAuthent
Dhcp6OptServerU
Dhcp6OptStatusC
Dhcp6OptRapidC
Dhcp6OptUserCla
Dhcp6OptVendorC
Dhcp6OptVendorI
Dhcp6OptInterfa

Dhcp6Opt
Dhcp6Opt
}
enum DHCP6_S
Dhcp6Sts
Dhcp6StsU
Dhcp6StsN
Dhcp6StsN
Dhcp6Sts
Dhcp6StsU
}
enum DHCP6_D
Dhcp6Dui
Dhcp6Dui
Dhcp6Dui
Dhcp6Dui

Functions

EFI_STATUS EFIAPI EfiDhcp6S
EFI_DHC
EFI_STATUS EFIAPI EfiDhcp6S
EFI_DHC
EFI_STATUS EFIAPI EfiDhcp6C
EFI_DHC
EFI_DHC
*Dhcp6Mo
EFI_DHC
*Dhcp6Co
EFI_STATUS EFIAPI EfiDhcp6C
EFI_DHC
EFI_DHC
*Dhcp6Cf
EFI_STATUS EFIAPI EfiDhcp6I
EFI_DHC
BOOLEA
EFI_DHC
*OptionRe
OptionCou
EFI_DHC
*OptionLi
EFI_DHC
*Retransm
TimeoutEv
EFI_DHC
ReplyCall
*Callback
EFI_STATUS EFIAPI EfiDhcp6I
EFI_DHC
BOOLEA
EFI_STATUS EFIAPI EfiDhcp6I
EFI_DHC
UINT32 A
EFI_IPv6_

EFI_STATUS EFIAPI EfiDhcp6Release (
EFI_DHCP6_PRO
UINT32 AddressC
EFI_IPv6_ADDRI
EFI_STATUS EFIAPI EfiDhcp6Parse (IN
EFI_DHCP6_PRO
EFI_DHCP6_PAC
OUT UINT32 *Op
EFI_DHCP6_PAC
*PacketOptionList

Variables

EFI_IPv6_ADDRESS mAllDhcpRelayAr
EFI_IPv6_ADDRESS mAllDhcpServers
EFI_DHCP6_PROTOCOL gDhcp6ProtocolTe

Detailed Description

Dhcp6 internal data structure and definition declaration.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Dhcp6SendConfirmMsg(), Dhcp6SendDeclineMsg(), Dhcp6SendInfoRequestMsg(), Dhcp6SendReleaseMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6SendRequestMsg(), and Dhcp6SendSolicitMsg().

#define DHCP6_CNF_IRT 1

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_CNF_MAX_DELAY 1

#define DHCP6_CNF_MRC 0

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_CNF_MRD 10

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_CNF_MRT 4

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_DEC_IRT 1

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_DEC_MRC 5

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_DEC_MRD 0

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_DEC_MRT 0

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_INF_IRT 1

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_INF_MAX_DELAY 1

#define DHCP6_INF_MRC 0

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_INF_MRD 0

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_INF_MRT 120

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_INSTANCE_FROM_THIS (Instance) CR ((Instance), DHCP6_INSTANCE, Dhcp6, DHCP6_INSTANCE_SIGNATURE)

Referenced by Dhcp6ServiceBindingDestroyChild(), EfiDhcp6Configure(), EfiDhcp6Decline(), EfiDhcp6GetModeData(), EfiDhcp6InfoRequest(), EfiDhcp6Release(), EfiDhcp6RenewRebind(), EfiDhcp6Start(), and EfiDhcp6Stop().

#define DHCP6_INSTANCE_SIGNATURE SIGNATURE_32 ('D', 'H', '6', 'I')

Referenced by Dhcp6CreateInstance(), and Dhcp6DestroyChildEntry().

#define DHCP6_PACKET_ALL 0

Referenced by Dhcp6CleanupRetry(), Dhcp6DestroyInstance(), and Dhcp6ReceivePacket().

#define DHCP6_PACKET_STATEFUL 1

Referenced by Dhcp6CleanupRetry(), and Dhcp6CleanupSession().

#define DHCP6_PACKET_STATELESS 2

Referenced by Dhcp6CleanupRetry(), and Dhcp6HandleStateless().

#define DHCP6_PORT_CLIENT 546

Referenced by Dhcp6ConfigureUdpIo(), and Dhcp6TransmitPacket().

#define DHCP6_PORT_SERVER 547

Referenced by Dhcp6TransmitPacket().

#define DHCP6_REB_IRT 10

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_REB_MRC 0

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_REB_MRD 0

#define DHCP6_REB_MRT 600

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_REL_IRT 1

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_REL_MRC 5

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_REL_MRD 0

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_REL_MRT 0

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_REN_IRT 10

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_REN_MRC 0

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_REN_MRD 0

#define DHCP6_REN_MRT 600

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_REQ_IRT 1

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_REQ_MRC 10

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_REQ_MRD 0

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_REQ_MRT 30

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_SERVICE_FROM_THIS (Service) CR ((Service), DHCP6_SERVICE, ServiceBinding, DHCP6_SERVICE_SIGNATURE)

Referenced by Dhcp6DriverBindingStop(), Dhcp6ServiceBindingCreateChild(), and Dhcp6ServiceBindingDestroyChild().

#define DHCP6_SERVICE_SIGNATURE SIGNATURE_32 ('D', 'H', '6', 'S')

Referenced by Dhcp6CreateService().

#define DHCP6_SOL_IRT 1

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_SOL_MAX_DELAY 1

#define DHCP6_SOL_MRC 0

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_SOL_MRD 0

Referenced by Dhcp6EnqueueRetry().

#define DHCP6_SOL_MRT 120

Referenced by Dhcp6EnqueueRetry().

Typedef Documentation

typedef struct _DHCP6_IA_CB DHCP6_IA_CB

typedef struct _DHCP6_INF_CB DHCP6_INF_CB

typedef struct _DHCP6_INSTANCE DHCP6_INSTANCE

typedef struct _DHCP6_SERVICE DHCP6_SERVICE

typedef struct _DHCP6_TX_CB DHCP6_TX_CB

Enumeration Type Documentation

enum DHCP6_DUID_TYPE

Enumerator:

Dhcp6DuidTypeLlt
Dhcp6DuidTypeEn
Dhcp6DuidTypeLl
Dhcp6DuidTypeUuid

enum DHCP6_MSG_TYPE

Enumerator:

Dhcp6MsgSolicit
Dhcp6MsgAdvertise
Dhcp6MsgRequest
Dhcp6MsgConfirm
Dhcp6MsgRenew
Dhcp6MsgRebind
Dhcp6MsgReply
Dhcp6MsgRelease
Dhcp6MsgDecline
Dhcp6MsgReconfigure
Dhcp6MsgInfoRequest

enum DHCP6_OPT_CODE

Enumerator:

Dhcp6OptClientId
Dhcp6OptServerId
Dhcp6OptIana
Dhcp6OptIata
Dhcp6OptIaAddr
Dhcp6OptRequestOption
Dhcp6OptPreference
Dhcp6OptElapsedTime
Dhcp6OptReplayMessage
Dhcp6OptAuthentication
Dhcp6OptServerUnicast

Dhcp6OptStatusCode
Dhcp6OptRapidCommit
Dhcp6OptUserClass
Dhcp6OptVendorClass
Dhcp6OptVendorInfo
Dhcp6OptInterfaceId
Dhcp6OptReconfigMessage
Dhcp6OptReconfigureAccept

enum DHCP6_STS_CODE

Enumerator:

Dhcp6StsSuccess
Dhcp6StsUnspecFail
Dhcp6StsNoAddrsAvail
Dhcp6StsNoBinding
Dhcp6StsNotOnLink
Dhcp6StsUseMulticast

Function Documentation

```
EFI_STATUS EFIAPI EfiDhcp6Configure ( IN EFI_DHCP6_PROTOCOL *        This,
                                      IN EFI_DHCP6_CONFIG_DATA *    Dhcp6CfgData
                                    )
```

Initializes, changes, or resets the operational settings for the Dhcp6 instance.

The Configure() function is used to initialize or clean up the configuration data of the Dhcp6 instance:

- When Dhcp6CfgData is not NULL and Configure() is called successfully, the configuration data will be initialized in the Dhcp6 instance and the state of the configured IA will be transferred into Dhcp6Init.
- When Dhcp6CfgData is NULL and Configure() is called successfully, the configuration data will be cleaned up and no IA will be associated with the Dhcp6 instance. To update the configuration data for an Dhcp6 instance, the original data must be cleaned up before setting the new configuration data.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol
[in]	*Dhcp6CfgData*	The pointer to the EFI_DHCP6_CONFIG_DATA.

Return values:

EFI_SUCCESS	The Dhcp6 is configured successfully with the Dhcp6Init state, or cleaned up the original configuration setting.
EFI_ACCESS_DENIED	The Dhcp6 instance has been already configured when Dhcp6CfgData is not NULL. The Dhcp6 instance has already started the DHCPv6 S.A.R.R when Dhcp6CfgData is NULL.
EFI_INVALID_PARAMETER	Some of the parameter is invalid.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

Initializes, changes, or resets the operational settings for the Dhcp6 instance.

The Configure() function is used to initialize or clean up the configuration data of the Dhcp6 instance:

- When Dhcp6CfgData is not NULL and Configure() is called successfully, the configuration data will be initialized in the Dhcp6 instance, and the state of the configured IA will be transferred into Dhcp6Init.
- When Dhcp6CfgData is NULL and Configure() is called successfully, the configuration data will be cleaned up and no IA will be associated with the Dhcp6 instance. To update the configuration data for an Dhcp6 instance, the original data must be cleaned up before setting the new configuration data.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol
[in]	*Dhcp6CfgData*	The pointer to the EFI_DHCP6_CONFIG_DATA.

Return values:

EFI_SUCCESS	The Dhcp6 is configured successfully with the Dhcp6Init state, or cleaned up the original configuration setting.
EFI_ACCESS_DENIED	The Dhcp6 instance was already configured. The Dhcp6 instance has already started the DHCPv6 S.A.R.R when Dhcp6CfgData is NULL.
EFI_INVALID_PARAMETER	Some of the parameter is invalid.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References _DHCP6_SERVICE::Child, _DHCP6_INSTANCE::Config, DHCP6_INSTANCE_FROM_THIS, Dhcp6CleanupConfigData(), Dhcp6CopyConfigData(), Dhcp6OptClientId, Dhcp6OptIana, Dhcp6OptIata, Dhcp6OptRapidCommit, Dhcp6OptReconfigureAccept, _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, and _DHCP6_INSTANCE::Service.

```
EFI_STATUS EFIAPI EfiDhcp6Decline ( IN EFI_DHCP6_PROTOCOL *  This,
                                    IN UINT32               AddressCount,
                                    IN EFI_IPv6_ADDRESS *   Addresses
                                  )
```

Inform that one or more addresses assigned by a server are already in use by another node.

The Decline() function is used to manually decline the assignment of IPv6 addresses, which have been already used by another node. If all IPv6 addresses of the configured IA are declined through this function, the state of the IA will switch through Dhcp6Declining to Dhcp6Init. Otherwise, the state of the IA will restore to Dhcp6Bound after the declining process. The Decline() can only be called when the IA is in Dhcp6Bound state. If the EFI_DHCP6_CONFIG_DATA.IaInfoEvent is NULL, this function is a blocking operation. It will return after the declining process finishes, or aborted by user.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol.
[in]	*AddressCount*	The number of declining addresses.
[in]	*Addresses*	The pointer to the buffer stored the declining addresses.

Return values:

EFI_SUCCESS	The DHCPv6 decline exchange process completed when EFI_DHCP6_CONFIG_DATA.IaInfoEvent was NULL. The Dhcp6 instance has sent Decline packet when EFI_DHCP6_CONFIG_DATA.IaInfoEvent is not NULL.

EFI_ACCESS_DENIED	The Dhcp6 instance hasn't been configured, or the state of the configured IA is not in Dhcp6Bound.
EFI_ABORTED	The DHCPv6 decline exchange process was aborted by the user.
EFI_NOT_FOUND	Any specified IPv6 address is not correlated with the configured IA for this instance.
EFI_INVALID_PARAMETER	Some parameter is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

Inform that one or more addresses assigned by a server are already in use by another node.

The Decline() function is used to manually decline the assignment of IPv6 addresses, which have been already used by another node. If all IPv6 addresses of the configured IA are declined through this function, the state of the IA will switch through Dhcp6Declining to Dhcp6Init. Otherwise, the state of the IA will restore to Dhcp6Bound after the declining process. The Decline() can only be called when the IA is in Dhcp6Bound state. If the EFI_DHCP6_CONFIG_DATA.IaInfoEvent is NULL, this function is a blocking operation. It will return after the declining process finishes, or aborted by user.

Parameters:

`[in]`	*This*	The pointer to EFI_DHCP6_PROTOCOL.
`[in]`	*AddressCount*	The number of declining addresses.
`[in]`	*Addresses*	The pointer to the buffer stored the declining addresses.

Return values:

EFI_SUCCESS	The DHCPv6 decline exchange process completed when EFI_DHCP6_CONFIG_DATA.IaInfoEvent was NULL. The Dhcp6 instance sent Decline packet when EFI_DHCP6_CONFIG_DATA.IaInfoEvent was not NULL.
EFI_ACCESS_DENIED	The Dhcp6 instance hasn't been configured, or the state of the configured IA is not in Dhcp6Bound.
EFI_ABORTED	The DHCPv6 decline exchange process aborted by user.
EFI_NOT_FOUND	Any specified IPv6 address is not correlated with the configured IA for this instance.
EFI_INVALID_PARAMETER	Some parameter is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References _DHCP6_INSTANCE::Config, DHCP6_INSTANCE_FROM_THIS, Dhcp6CheckAddress(), Dhcp6DepriveAddress(), Dhcp6ReceivePacket(), Dhcp6SendDeclineMsg(), _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, _DHCP6_INSTANCE::Service, _DHCP6_SERVICE::UdpIo, and _DHCP6_INSTANCE::UdpSts.

```
EFI_STATUS EFIAPI
EfiDhcp6GetModeData
                    ( IN EFI_DHCP6_PROTOCOL *          This,
                      OUT EFI_DHCP6_MODE_DATA *        Dhcp6ModeData,
                      OUT EFI_DHCP6_CONFIG_DATA *      Dhcp6ConfigData
                    )
```

Returns the current operating mode data for the Dhcp6 instance.

The GetModeData() function returns the current operating mode and cached data packet for the Dhcp6 instance.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol.
[out]	*Dhcp6ModeData*	The pointer to the Dhcp6 mode data.
[out]	*Dhcp6ConfigData*	The pointer to the Dhcp6 configure data.

Return values:

EFI_SUCCESS	The mode data was returned.
EFI_INVALID_PARAMETER	This is NULL.
EFI_ACCESS_DENIED	The EFI DHCPv6 Protocol instance has not been configured when Dhcp6ConfigData is not NULL.

Returns the current operating mode data for the Dhcp6 instance.

The GetModeData() function returns the current operating mode and cached data packet for the Dhcp6 instance.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol.
[out]	*Dhcp6ModeData*	The pointer to the Dhcp6 mode data.
[out]	*Dhcp6ConfigData*	The pointer to the Dhcp6 configure data.

Return values:

EFI_SUCCESS	The mode data was returned.
EFI_INVALID_PARAMETER	This is NULL.
EFI_ACCESS_DENIED	The EFI DHCPv6 Protocol instance was not configured.

References _DHCP6_SERVICE::ClientId, _DHCP6_INSTANCE::Config, DHCP6_INSTANCE_FROM_THIS, Dhcp6CleanupConfigData(), Dhcp6CleanupModeData(), Dhcp6CopyConfigData(), _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, and _DHCP6_INSTANCE::Service.

```
EFI_STATUS EFIAPI EfiDhcp6InfoRequest (
    IN EFI_DHCP6_PROTOCOL *                 This,
    IN BOOLEAN                              SendClientId,
    IN EFI_DHCP6_PACKET_OPTION *            OptionRequest,
    IN UINT32                               OptionCount,
    IN EFI_DHCP6_PACKET_OPTION *            OptionList[],
    IN EFI_DHCP6_RETRANSMISSION *           Retransmission,
    IN EFI_EVENT                            TimeoutEvent,
    IN EFI_DHCP6_INFO_CALLBACK              ReplyCallback,
    IN VOID *                               CallbackContext
)
```

Request configuration information without the assignment of any Ia addresses of the client.

The InfoRequest() function is used to request configuration information without the assignment of any IPv6 address of the client. Client sends out Information Request packet to obtain the required configuration information, and DHCPv6 server responds with Reply packet containing the information for the client. The

received Reply packet will be passed to the user by ReplyCallback function. If user returns EFI_NOT_READY from ReplyCallback, the Dhcp6 instance will continue to receive other Reply packets unless timeout according to the Retransmission parameter. Otherwise, the Information Request exchange process will be finished successfully if user returns EFI_SUCCESS from ReplyCallback.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol.
[in]	*SendClientId*	If TRUE, the DHCPv6 protocol instance will build Client Identifier option and include it into Information Request packet. Otherwise, Client Identifier option will not be included.
[in]	*OptionRequest*	The pointer to the buffer of option request options.
[in]	*OptionCount*	The option number in the OptionList.
[in]	*OptionList*	The list of appended options.
[in]	*Retransmission*	The pointer to the retransmission of the message.
[in]	*TimeoutEvent*	The event of timeout.
[in]	*ReplyCallback*	The callback function when a reply was received.
[in]	*CallbackContext*	The pointer to the parameter passed to the callback.

Return values:

EFI_SUCCESS	The DHCPv6 information request exchange process completed when TimeoutEvent is NULL. Information Request packet has been sent to DHCPv6 server when TimeoutEvent is not NULL.
EFI_NO_RESPONSE	The DHCPv6 information request exchange process failed because of no response, or not all requested-options are responded to by DHCPv6 servers when Timeout happened.
EFI_ABORTED	The DHCPv6 information request exchange process was aborted by the user.
EFI_INVALID_PARAMETER	Some parameter is NULL.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

Request configuration information without the assignment of any Ia addresses of the client.

The InfoRequest() function is used to request configuration information without the assignment of any IPv6 address of the client. The client sends out an Information Request packet to obtain the required configuration information, and DHCPv6 server responds with a Reply packet containing the information for the client. The received Reply packet will be passed to the user by ReplyCallback function. If the user returns EFI_NOT_READY from ReplyCallback, the Dhcp6 instance will continue to receive other Reply packets unless timeout according to the Retransmission parameter. Otherwise, the Information Request exchange process will be finished successfully if user returns EFI_SUCCESS from ReplyCallback.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol.
[in]	*SendClientId*	If TRUE, the DHCPv6 protocol instance will build Client Identifier option and include it into Information Request packet. Otherwise, Client Identifier option will not be included.
[in]	*OptionRequest*	The pointer to the buffer of option request options.
[in]	*OptionCount*	The option number in the OptionList.
[in]	*OptionList*	The list of appended options.
[in]	*Retransmission*	The pointer to the retransmission of the message.
[in]	*TimeoutEvent*	The event of timeout.
[in]	*ReplyCallback*	The callback function when the reply was received.

[in] *CallbackContext* The pointer to the parameter passed to the callback.

Return values:

EFI_SUCCESS	The DHCPv6 information request exchange process completed when TimeoutEvent is NULL. Information Request packet has been sent to DHCPv6 server when TimeoutEvent is not NULL.
EFI_NO_RESPONSE	The DHCPv6 information request exchange process failed because of no response, or not all requested-options are responded by DHCPv6 servers when Timeout happened.
EFI_ABORTED	The DHCPv6 information request exchange process was aborted by user.
EFI_INVALID_PARAMETER	Some parameter is NULL.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References DHCP6_INSTANCE_FROM_THIS, Dhcp6GetMappingTimeOut(), Dhcp6OptClientId, Dhcp6OptRequestOption, Dhcp6StartInfoRequest(), _DHCP6_SERVICE::Ip6Cfg, _DHCP6_INSTANCE::Service, _DHCP6_SERVICE::UdpIo, and _DHCP6_INSTANCE::UdpSts.

```
EFI_STATUS EFIAPI EfiDhcp6Parse ( IN EFI_DHCP6_PROTOCOL *          This,
                                  IN EFI_DHCP6_PACKET *             Packet,
                                  IN OUT UINT32 *                   OptionCount,
                                  OUT EFI_DHCP6_PACKET_OPTION *     PacketOptionList[]
                                  )
```

Parse the option data in the Dhcp6 packet.

The Parse() function is used to retrieve the option list in the DHCPv6 packet.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol.
[in]	*Packet*	The pointer to the Dhcp6 packet.
[in,out]	*OptionCount*	The number of option in the packet.
[out]	*PacketOptionList*	The array of pointers to the each option in the packet.

Return values:

EFI_SUCCESS	The packet was successfully parsed.
EFI_INVALID_PARAMETER	Some parameter is NULL.
EFI_BUFFER_TOO_SMALL	*OptionCount is smaller than the number of options that were found in the Packet.

Parse the option data in the Dhcp6 packet.

The Parse() function is used to retrieve the option list in the DHCPv6 packet.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol.
[in]	*Packet*	The pointer to the Dhcp6 packet.
[in,out]	*OptionCount*	The number of option in the packet.
[out]	*PacketOptionList*	The array of pointers to each option in the packet.

Return values:

EFI_SUCCESS	The packet was successfully parsed.
EFI_INVALID_PARAMETER	Some parameter is NULL.
EFI_BUFFER_TOO_SMALL	*OptionCount is smaller than the number of options that were found in the Packet.

```
EFI_STATUS EFIAPI EfiDhcp6Release ( IN EFI_DHCP6_PROTOCOL *  This,
                                    IN UINT32               AddressCount,
                                    IN EFI_IPv6_ADDRESS *   Addresses
                                  )
```

Release one or more addresses associated with the configured Ia for the current instance.

The Release() function is used to manually release the one or more IPv6 address. If AddressCount is zero, it will release all IPv6 addresses of the configured IA. If all IPv6 addresses of the IA are released through this function, the state of the IA will switch through Dhcp6Releasing to Dhcp6Init, otherwise, the state of the IA will restore to Dhcp6Bound after the releasing process. The Release() can only be called when the IA is in a Dhcp6Bound state. If the EFI_DHCP6_CONFIG_DATA.IaInfoEvent is NULL, the function is a blocking operation. It will return after the releasing process finishes, or aborted by user.

Parameters:

`[in]`	*This*	The pointer to the Dhcp6 protocol.
`[in]`	*AddressCount*	The number of releasing addresses.
`[in]`	*Addresses*	The pointer to the buffer stored the releasing addresses.

Return values:

EFI_SUCCESS	The DHCPv6 release exchange process has completed when EFI_DHCP6_CONFIG_DATA.IaInfoEvent is NULL. The Dhcp6 instance has sent Release packet when EFI_DHCP6_CONFIG_DATA.IaInfoEvent is not NULL.
EFI_ACCESS_DENIED	The Dhcp6 instance hasn't been configured, or the state of the configured IA is not in Dhcp6Bound.
EFI_ABORTED	The DHCPv6 release exchange process was aborted by the user.
EFI_NOT_FOUND	Any specified IPv6 address is not correlated with the configured IA for this instance.
EFI_INVALID_PARAMETER	Some parameter is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

Release one or more addresses associated with the configured Ia for current instance.

The Release() function is used to manually release one or more IPv6 addresses. If AddressCount is zero, it will release all IPv6 addresses of the configured IA. If all IPv6 addresses of the IA are released through this function, the state of the IA will switch through Dhcp6Releasing to Dhcp6Init, otherwise, the state of the IA will restore to Dhcp6Bound after the releasing process. The Release() can only be called when the IA is in Dhcp6Bound state. If the EFI_DHCP6_CONFIG_DATA.IaInfoEvent is NULL, the function is a blocking operation. It will return after the releasing process finishes, or is aborted by user.

Parameters:

`[in]`	*This*	The pointer to the Dhcp6 protocol.
`[in]`	*AddressCount*	The number of releasing addresses.
`[in]`	*Addresses*	The pointer to the buffer stored the releasing addresses.

Return values:

EFI_SUCCESS	The DHCPv6 release exchange process completed when EFI_DHCP6_CONFIG_DATA.IaInfoEvent was NULL. The Dhcp6 instance was sent Release packet when EFI_DHCP6_CONFIG_DATA.IaInfoEvent was not NULL.
EFI_ACCESS_DENIED	The Dhcp6 instance hasn't been configured, or the state of the configured IA is not in Dhcp6Bound.
EFI_ABORTED	The DHCPv6 release exchange process aborted by user.
EFI_NOT_FOUND	Any specified IPv6 address is not correlated with the configured IA for this instance.
EFI_INVALID_PARAMETER	Some parameter is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References _DHCP6_INSTANCE::Config, DHCP6_INSTANCE_FROM_THIS, Dhcp6CheckAddress(), Dhcp6DepriveAddress(), Dhcp6ReceivePacket(), Dhcp6SendReleaseMsg(), _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, _DHCP6_INSTANCE::Service, _DHCP6_SERVICE::UdpIo, and _DHCP6_INSTANCE::UdpSts.

```
EFI_STATUS EFIAPI EfiDhcp6RenewRebind ( IN EFI_DHCP6_PROTOCOL *  This,
                                        IN BOOLEAN               RebindRequest
                                      )
```

Manually extend the valid and preferred lifetimes for the IPv6 addresses of the configured IA and update other configuration parameters by sending Renew or Rebind packet.

The RenewRebind() function is used to manually extend the valid and preferred lifetimes for the IPv6 addresses of the configured IA and update other configuration parameters by sending a Renew or Rebind packet.

- When RebindRequest is FALSE and the state of the configured IA is Dhcp6Bound, it will send Renew packet to the previously DHCPv6 server and transfer the state of the configured IA to Dhcp6Renewing. If valid Reply packet received, the state transfers to Dhcp6Bound and the valid and preferred timer restarts. If fails, the state transfers to Dhcp6Bound but the timer continues.
- When RebindRequest is TRUE and the state of the configured IA is Dhcp6Bound, it will send a Rebind packet. If a valid Reply packet is received, the state transfers to Dhcp6Bound, and the valid and preferred timer restarts. If it fails, the state transfers to Dhcp6Init, and the IA can't be used.

Parameters:

[in]	*This*	The pointer to the Dhcp6 protocol.
[in]	*RebindRequest*	If TRUE, Rebind packet will be sent and enter Dhcp6Rebinding state. Otherwise, Renew packet will be sent and enter Dhcp6Renewing state.

Return values:

EFI_SUCCESS	The DHCPv6 renew/rebind exchange process completed and at least one IPv6 address of the configured IA was bound again when EFI_DHCP6_CONFIG_DATA.IaInfoEvent was NULL. The EFI DHCPv6 Protocol instance has sent Renew or Rebind packet when EFI_DHCP6_CONFIG_DATA.IaInfoEvent is not NULL.
EFI_ACCESS_DENIED	The Dhcp6 instance hasn't been configured, or the state of the configured IA is not in Dhcp6Bound.
EFI_ALREADY_STARTED	The state of the configured IA has already entered Dhcp6Renewing when RebindRequest is FALSE. The state of the configured IA has already entered Dhcp6Rebinding when RebindRequest is TRUE.
EFI_ABORTED	The DHCPv6 renew/rebind exchange process aborted by user.

EFI_NO_RESPONSE	The DHCPv6 renew/rebind exchange process failed because of no response.
EFI_NO_MAPPING	No IPv6 address has been bound to the configured IA after the DHCPv6 renew/rebind exchange process.
EFI_INVALID_PARAMETER	Some parameter is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

Manually extend the valid and preferred lifetimes for the IPv6 addresses of the configured IA and update other configuration parameters by sending a Renew or Rebind packet.

The RenewRebind() function is used to manually extend the valid and preferred lifetimes for the IPv6 addresses of the configured IA, and update other configuration parameters by sending Renew or Rebind packet.

- When RebindRequest is FALSE and the state of the configured IA is Dhcp6Bound, it sends Renew packet to the previously DHCPv6 server and transfer the state of the configured IA to Dhcp6Renewing. If valid Reply packet received, the state transfers to Dhcp6Bound and the valid and preferred timer restarts. If fails, the state transfers to Dhcp6Bound, but the timer continues.
- When RebindRequest is TRUE and the state of the configured IA is Dhcp6Bound, it will send a Rebind packet. If valid Reply packet is received, the state transfers to Dhcp6Bound and the valid and preferred timer restarts. If it fails, the state transfers to Dhcp6Init, and the IA can't be used.

Parameters:

`[in]`	*This*	The pointer to the Dhcp6 protocol.
`[in]`	*RebindRequest*	If TRUE, Rebind packet will be sent and enter Dhcp6Rebinding state. Otherwise, Renew packet will be sent and enter Dhcp6Renewing state.

Return values:

EFI_SUCCESS	The DHCPv6 renew/rebind exchange process has completed and at least one IPv6 address of the configured IA has been bound again when EFI_DHCP6_CONFIG_DATA.IaInfoEvent is NULL. The EFI DHCPv6 Protocol instance has sent Renew or Rebind packet when EFI_DHCP6_CONFIG_DATA.IaInfoEvent is not NULL.
EFI_ACCESS_DENIED	The Dhcp6 instance hasn't been configured, or the state of the configured IA is not in Dhcp6Bound.
EFI_ALREADY_STARTED	The state of the configured IA has already entered Dhcp6Renewing when RebindRequest is FALSE. The state of the configured IA has already entered Dhcp6Rebinding when RebindRequest is TRUE.
EFI_ABORTED	The DHCPv6 renew/rebind exchange process aborted by the user.
EFI_NO_RESPONSE	The DHCPv6 renew/rebind exchange process failed because of no response.
EFI_NO_MAPPING	No IPv6 address has been bound to the configured IA after the DHCPv6 renew/rebind exchange process.
EFI_INVALID_PARAMETER	Some parameter is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References _DHCP6_INSTANCE::Config, DHCP6_INSTANCE_FROM_THIS, Dhcp6ReceivePacket(), Dhcp6SendRenewRebindMsg(), _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, _DHCP6_INSTANCE::Service, _DHCP6_SERVICE::UdpIo, and _DHCP6_INSTANCE::UdpSts.

EFI_STATUS EFIAPI EfiDhcp6Start (IN EFI_DHCP6_PROTOCOL * *This*)

Starts the DHCPv6 standard S.A.R.R. process.

The Start() function starts the DHCPv6 standard process. This function can be called only when the state of Dhcp6 instance is in the Dhcp6Init state. If the DHCP process completes successfully, the state of the Dhcp6 instance will be transferred through Dhcp6Selecting and Dhcp6Requesting to the Dhcp6Bound state. Refer to rfc-3315 for precise state transitions during this process. At the time when each event occurs in this process, the callback function that was set by EFI_DHCP6_PROTOCOL.Configure() will be called and the user can take this opportunity to control the process.

Parameters:

`[in]` *This* The pointer to Dhcp6 protocol.

Return values:

EFI_SUCCESS	The DHCPv6 standard process has started, or it completed when CompletionEvent was NULL.
EFI_ACCESS_DENIED	The EFI DHCPv6 Child instance hasn't been configured.
EFI_INVALID_PARAMETER	This is NULL.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_TIMEOUT	The DHCPv6 configuration process failed because no response was received from the server within the specified timeout value.
EFI_ABORTED	The user aborted the DHCPv6 process.
EFI_ALREADY_STARTED	Some other Dhcp6 instance already started the DHCPv6 standard process.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

Starts the DHCPv6 standard S.A.R.R. process.

The Start() function starts the DHCPv6 standard process. This function can be called only when the state of Dhcp6 instance is in the Dhcp6Init state. If the DHCP process completes successfully, the state of the Dhcp6 instance will be transferred through Dhcp6Selecting and Dhcp6Requesting to the Dhcp6Bound state. Refer to rfc-3315 for precise state transitions during this process. At the time when each event occurs in this process, the callback function that was set by EFI_DHCP6_PROTOCOL.Configure() will be called, and the user can take this opportunity to control the process.

Parameters:

`[in]` *This* The pointer to Dhcp6 protocol.

Return values:

EFI_SUCCESS	The DHCPv6 standard process has started, or it has completed when CompletionEvent is NULL.
EFI_ACCESS_DENIED	The EFI DHCPv6 Child instance hasn't been configured.
EFI_INVALID_PARAMETER	This is NULL.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_TIMEOUT	The DHCPv6 configuration process failed because no response was received from the server within the specified timeout value.
EFI_ABORTED	The user aborted the DHCPv6 process.
EFI_ALREADY_STARTED	Some other Dhcp6 instance already started the DHCPv6 standard process.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_NO_MEDIA	There was a media error.

References _DHCP6_INSTANCE::Config, DHCP6_INSTANCE_FROM_THIS, Dhcp6ReceivePacket(), Dhcp6SendSolicitMsg(), _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, _DHCP6_INSTANCE::Service, _DHCP6_SERVICE::UdpIo, and _DHCP6_INSTANCE::UdpSts.

EFI_STATUS EFIAPI EfiDhcp6Stop (IN EFI_DHCP6_PROTOCOL * *This*)
Stops the DHCPv6 standard S.A.R.R. process.

The Stop() function is used to stop the DHCPv6 standard process. After this function is called successfully, the state of Dhcp6 instance is transferred into the Dhcp6Init. EFI_DHCP6_PROTOCOL.Configure() needs to be called before DHCPv6 standard process can be started again. This function can be called when the Dhcp6 instance is in any state.

Parameters:

`[in]` *This* The pointer to the Dhcp6 protocol.

Return values:

EFI_SUCCESS	The Dhcp6 instance is now in the Dhcp6Init state.
EFI_INVALID_PARAMETER	This is NULL.

Stops the DHCPv6 standard S.A.R.R. process.

The Stop() function is used to stop the DHCPv6 standard process. After this function is called successfully, the state of Dhcp6 instance is transferred into Dhcp6Init. EFI_DHCP6_PROTOCOL.Configure() needs to be called before DHCPv6 standard process can be started again. This function can be called when the Dhcp6 instance is in any state.

Parameters:

`[in]` *This* The pointer to the Dhcp6 protocol.

Return values:

EFI_SUCCESS	The Dhcp6 instance is now in the Dhcp6Init state.
EFI_INVALID_PARAMETER	This is NULL.

References _DHCP6_INSTANCE::Config, DHCP6_INSTANCE_FROM_THIS, Dhcp6CleanupSession(), Dhcp6SendReleaseMsg(), _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, _DHCP6_INSTANCE::Service, _DHCP6_SERVICE::UdpIo, and _DHCP6_INSTANCE::UdpSts.

Variable Documentation

EFI_DHCP6_PROTOCOL gDhcp6ProtocolTemplate

Referenced by Dhcp6CreateInstance().

EFI_IPv6_ADDRESS mAllDhcpRelayAndServersAddress

Referenced by Dhcp6TransmitPacket(), and PxeBcRetryDhcp6Binl().

EFI_IPv6_ADDRESS mAllDhcpServersAddress

- File List
- Globals

NetworkPkg/Dhcp6Dxe/Dhcp6Io.c File Reference

Functions

EFI_STATUS Dhcp6EnqueueRetry (IN DHCP6_INSTANCE *Instance, IN EFI_DHCP6_PACKET *Packet, IN UINT16 *Elapsed, IN EFI_DHCP6_RETRANSMISSION *RetryCtl)
EFI_STATUS Dhcp6DequeueRetry (IN DHCP6_INSTANCE *Instance, IN UINT32 PacketXid, IN BOOLEAN NeedSignal)
VOID Dhcp6CleanupRetry (IN DHCP6_INSTANCE *Instance, IN UINT32 Scope)
BOOLEAN Dhcp6IsValidTxCb (IN DHCP6_INSTANCE *Instance, IN DHCP6_TX_CB *TxCb)
VOID Dhcp6CleanupSession (IN OUT DHCP6_INSTANCE *Instance, IN EFI_STATUS Status)
EFI_STATUS EFIAPI Dhcp6CallbackUser (IN DHCP6_INSTANCE *Instance, IN EFI_DHCP6_EVENT Event, IN OUT EFI_DHCP6_PACKET **Packet)
EFI_STATUS Dhcp6UpdateIaInfo (IN OUT DHCP6_INSTANCE *Instance, IN EFI_DHCP6_PACKET *Packet)
EFI_STATUS Dhcp6SeekStsOption (IN DHCP6_INSTANCE *Instance, IN EFI_DHCP6_PACKET *Packet, OUT UINT8 **Option)
EFI_STATUS Dhcp6TransmitPacket (IN DHCP6_INSTANCE *Instance, IN EFI_DHCP6_PACKET *Packet, IN UINT16 *Elapsed)
EFI_STATUS Dhcp6SendSolicitMsg (IN DHCP6_INSTANCE *Instance)
EFI_STATUS Dhcp6InitSolicitMsg (IN DHCP6_INSTANCE *Instance)
EFI_STATUS Dhcp6SendRequestMsg (IN DHCP6_INSTANCE *Instance)
EFI_STATUS Dhcp6SendDeclineMsg (IN DHCP6_INSTANCE *Instance, IN EFI_DHCP6_IA *DecIa)
EFI_STATUS Dhcp6SendReleaseMsg (IN DHCP6_INSTANCE *Instance, IN EFI_DHCP6_IA *RelIa)
EFI_STATUS Dhcp6SendRenewRebindMsg (IN DHCP6_INSTANCE *Instance, IN BOOLEAN RebindRequest)
EFI_STATUS Dhcp6StartInfoRequest (IN DHCP6_INSTANCE *Instance, IN BOOLEAN SendClientId, IN EFI_DHCP6_PACKET_OPTION *OptionRequest, IN UINT32 OptionCount, IN EFI_DHCP6_PACKET_OPTION *OptionList[], IN EFI_DHCP6_RETRANSMISSION *Retransmission, IN EFI_EVENT TimeoutEvent, IN EFI_DHCP6_INFO_CALLBACK ReplyCallback, IN VOID *CallbackContext)
EFI_STATUS Dhcp6SendInfoRequestMsg (IN DHCP6_INSTANCE *Instance, IN DHCP6_INF_CB *InfCb, IN BOOLEAN SendClientId, IN EFI_DHCP6_PACKET_OPTION *OptionRequest, IN UINT32 OptionCount, IN EFI_DHCP6_PACKET_OPTION *OptionList[], IN EFI_DHCP6_RETRANSMISSION *Retransmission)
EFI_STATUS Dhcp6SendConfirmMsg (IN DHCP6_INSTANCE *Instance)
EFI_STATUS Dhcp6HandleReplyMsg (IN DHCP6_INSTANCE *Instance, IN EFI_DHCP6_PACKET *Packet)
EFI_STATUS Dhcp6SelectAdvertiseMsg (IN DHCP6_INSTANCE *Instance, IN EFI_DHCP6_PACKET *AdSelect)
EFI_STATUS Dhcp6HandleAdvertiseMsg (IN DHCP6_INSTANCE *Instance, IN EFI_DHCP6_PACKET *Packet)
VOID Dhcp6HandleStateful (IN DHCP6_INSTANCE *Instance, IN EFI_DHCP6_PACKET *Packet)
VOID Dhcp6HandleStateless (IN DHCP6_INSTANCE *Instance, IN EFI_DHCP6_PACKET *Packet)
VOID EFIAPI Dhcp6ReceivePacket (IN NET_BUF *Udp6Wrap, IN UDP_END_POINT *EndPoint, IN EFI_STATUS IoStatus, IN VOID *Context)

BOOLEAN Dhcp6LinkMovDetect (IN DHCP6_INSTANCE *Instance)
VOID EFIAPI Dhcp6OnTimerTick (IN EFI_EVENT Event, IN VOID *Context)

Detailed Description

Dhcp6 internal functions implementation.

(C) Copyright 2014 Hewlett-Packard Development Company, L.P.
Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI Dhcp6CallbackUser ( IN DHCP6_INSTANCE *          Instance,
                                      IN EFI_DHCP6_EVENT           Event,
                                      IN OUT EFI_DHCP6_PACKET **   Packet
                                    )
```

Callback to user when Dhcp6 transmit/receive occurs.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*Event*	The current Dhcp6 event.
[in,out]	*Packet*	The pointer to the packet sending or received.

Return values:

EFI_SUCCESS	The user function returns success.
EFI_NOT_READY	Direct the caller to continue collecting the offer.
EFI_ABORTED	The user function ask it to abort.

Referenced by Dhcp6HandleAdvertiseMsg(), Dhcp6HandleReplyMsg(), Dhcp6SelectAdvertiseMsg(), Dhcp6SendConfirmMsg(), Dhcp6SendDeclineMsg(), Dhcp6SendReleaseMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6SendRequestMsg(), and Dhcp6SendSolicitMsg().

```
VOID Dhcp6CleanupRetry ( IN DHCP6_INSTANCE *   Instance,
                         IN UINT32             Scope
                       )
```

Clean up the specific nodes in the retry list.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*Scope*	The scope of cleanup nodes.

References DHCP6_PACKET_ALL, DHCP6_PACKET_STATEFUL, DHCP6_PACKET_STATELESS, Dhcp6MsgInfoRequest, _DHCP6_INF_CB::Link, _DHCP6_TX_CB::Link, _DHCP6_INF_CB::TimeoutEvent, and _DHCP6_TX_CB::TxPacket.

Referenced by Dhcp6CleanupSession(), Dhcp6DestroyInstance(), Dhcp6HandleStateless(), and Dhcp6ReceivePacket().

```
VOID Dhcp6CleanupSession ( IN OUT DHCP6_INSTANCE *   Instance,
                           IN EFI_STATUS             Status
                         )
```

Clean up the session of the instance stateful exchange.

Parameters:

[in,out]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*Status*	The return status from udp.

References DHCP6_PACKET_STATEFUL, and Dhcp6CleanupRetry().

Referenced by Dhcp6HandleStateful(), Dhcp6OnTimerTick(), and EfiDhcp6Stop().

```
EFI_STATUS Dhcp6DequeueRetry ( IN DHCP6_INSTANCE *   Instance,
                               IN UINT32             PacketXid,
                               IN BOOLEAN            NeedSignal
                             )
```

Dequeue the packet from retry list if reply received or timeout at last.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*PacketXid*	The packet transaction id to match.
[in]	*NeedSignal*	If TRUE, then an timeout event need be signaled when it is existed. Otherwise, this parameter is ignored.

Return values:

EFI_SUCCESS	Successfully dequeued the packet into retry list .
EFI_NOT_FOUND	There is no xid matched in retry list.

References Dhcp6MsgInfoRequest, _DHCP6_TX_CB::Link, _DHCP6_INF_CB::Link, _DHCP6_INF_CB::TimeoutEvent, _DHCP6_TX_CB::TxPacket, _DHCP6_INF_CB::Xid, and _DHCP6_TX_CB::Xid.

Referenced by Dhcp6HandleReplyMsg(), Dhcp6HandleStateless(), Dhcp6OnTimerTick(), and Dhcp6SelectAdvertiseMsg().

```
EFI_STATUS Dhcp6EnqueueRetry ( IN DHCP6_INSTANCE *                Instance,
                               IN EFI_DHCP6_PACKET *              Packet,
                               IN UINT16 *                        Elapsed,
                               IN EFI_DHCP6_RETRANSMISSION *      RetryCtl
                             )
```

Enqueue the packet into the retry list in case of timeout.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*Packet*	The pointer to the Dhcp6 packet to retry.
[in]	*Elapsed*	The pointer to the elapsed time value in the packet.
[in]	*RetryCtl*	The pointer to the transmission control of the packet. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	Successfully enqueued the packet into the retry list according to its message type.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected message type.

References _DHCP6_IA_CB::AllExpireTime, DHCP6_CNF_IRT, DHCP6_CNF_MRC, DHCP6_CNF_MRD, DHCP6_CNF_MRT, DHCP6_DEC_IRT, DHCP6_DEC_MRC, DHCP6_DEC_MRD, DHCP6_DEC_MRT, DHCP6_INF_IRT, DHCP6_INF_MRC, DHCP6_INF_MRD, DHCP6_INF_MRT, DHCP6_REB_IRT, DHCP6_REB_MRC, DHCP6_REB_MRT, DHCP6_REL_IRT, DHCP6_REL_MRC, DHCP6_REL_MRD, DHCP6_REL_MRT, DHCP6_REN_IRT, DHCP6_REN_MRC, DHCP6_REN_MRT, DHCP6_REQ_IRT, DHCP6_REQ_MRC, DHCP6_REQ_MRD, DHCP6_REQ_MRT, DHCP6_SOL_IRT, DHCP6_SOL_MRC, DHCP6_SOL_MRD, DHCP6_SOL_MRT, Dhcp6CalculateExpireTime(), Dhcp6MsgConfirm, Dhcp6MsgDecline, Dhcp6MsgInfoRequest, Dhcp6MsgRebind, Dhcp6MsgRelease, Dhcp6MsgRenew, Dhcp6MsgRequest, Dhcp6MsgSolicit, _DHCP6_TX_CB::Elapsed, _DHCP6_TX_CB::Link, _DHCP6_TX_CB::RetryCtl, _DHCP6_TX_CB::RetryExp, _DHCP6_IA_CB::T1, _DHCP6_IA_CB::T2, _DHCP6_TX_CB::TxPacket, and _DHCP6_TX_CB::Xid.

Referenced by Dhcp6SendConfirmMsg(), Dhcp6SendDeclineMsg(), Dhcp6SendInfoRequestMsg(), Dhcp6SendReleaseMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6SendRequestMsg(), and Dhcp6SendSolicitMsg().

```
EFI_STATUS Dhcp6HandleAdvertiseMsg ( IN DHCP6_INSTANCE *      Instance,
                                     IN EFI_DHCP6_PACKET *    Packet
                                   )
```

Handle with the Dhcp6 advertisement message.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*Packet*	The pointer to the Dhcp6 advertisement message.

Return values:

EFI_SUCCESS	Processed the advertisement message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.
Others	Failed to process the advertise message.

References Dhcp6CallbackUser(), Dhcp6HandleReplyMsg(), Dhcp6MsgAdvertise, Dhcp6MsgReply, Dhcp6OptPreference, Dhcp6OptRapidCommit, Dhcp6SeekOption(), Dhcp6SeekStsOption(), and Dhcp6SelectAdvertiseMsg().

Referenced by Dhcp6HandleStateful().

```
EFI_STATUS Dhcp6HandleReplyMsg ( IN DHCP6_INSTANCE *      Instance,
                                 IN EFI_DHCP6_PACKET *    Packet
                               )
```

Handle with the Dhcp6 reply message.

Parameters:

[in]	*Instance*	The pointer to Dhcp6 instance.
[in]	*Packet*	The pointer to the Dhcp6 reply message.

Return values:

EFI_SUCCESS	Processed the reply message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.
Others	Failed to process the reply message.

References Dhcp6AppendCacheIa(), Dhcp6CacheIa(), Dhcp6CallbackUser(), Dhcp6DequeueRetry(), Dhcp6InitSolicitMsg(), Dhcp6MsgReply, Dhcp6OptRapidCommit, Dhcp6SeekIaOption(), Dhcp6SeekOption(), Dhcp6SeekStsOption(), Dhcp6SendRequestMsg(), Dhcp6StsNoBinding, Dhcp6StsNotOnLink, Dhcp6StsUnspecFail, Dhcp6StsUseMulticast, and Dhcp6UpdateIaInfo().

Referenced by Dhcp6HandleAdvertiseMsg(), and Dhcp6HandleStateful().

```
VOID Dhcp6HandleStateful ( IN DHCP6_INSTANCE *      Instance,
                           IN EFI_DHCP6_PACKET *    Packet
                         )
```

The Dhcp6 stateful exchange process routine.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*Packet*	The pointer to the received Dhcp6 message.

References _DHCP6_SERVICE::ClientId, Dhcp6CleanupSession(), Dhcp6HandleAdvertiseMsg(), Dhcp6HandleReplyMsg(), Dhcp6MsgAdvertise, Dhcp6MsgReply, Dhcp6OptClientId, Dhcp6OptServerId, Dhcp6ReceivePacket(), Dhcp6SeekOption(), and _DHCP6_SERVICE::UdpIo.

Referenced by Dhcp6ReceivePacket().

```
VOID Dhcp6HandleStateless ( IN DHCP6_INSTANCE *      Instance,
                            IN EFI_DHCP6_PACKET *    Packet
                          )
```

The Dhcp6 stateless exchange process routine.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*Packet*	The pointer to the received Dhcp6 message.

References _DHCP6_INF_CB::CallbackContext, DHCP6_PACKET_STATELESS, Dhcp6CleanupRetry(), Dhcp6DequeueRetry(), Dhcp6MsgReply, Dhcp6OptServerId, Dhcp6ReceivePacket(), Dhcp6SeekOption(), _DHCP6_INF_CB::ReplyCallback, _DHCP6_SERVICE::UdpIo, and _DHCP6_INF_CB::Xid.

Referenced by Dhcp6ReceivePacket().

EFI_STATUS Dhcp6InitSolicitMsg (IN DHCP6_INSTANCE * *Instance*)

Configure some parameter to initiate SolicitMsg.

Parameters:

[in] *Instance* The pointer to the Dhcp6 instance.

Return values:

EFI_SUCCESS	Created and sent the solicit message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
Others	Failed to send the solicit message.

References Dhcp6SendSolicitMsg().

Referenced by Dhcp6HandleReplyMsg().

BOOLEAN Dhcp6IsValidTxCb (IN DHCP6_INSTANCE * *Instance*,
IN DHCP6_TX_CB * *TxCb*
)

Check whether the TxCb is still a valid control block in the instance's retry list.

Parameters:

[in] *Instance* The pointer to DHCP6_INSTANCE.

[in] *TxCb* The control block for a transmitted message.

Return values:

TRUE The control block is in Instance's retry list.

FALSE The control block is NOT in Instance's retry list.

Referenced by Dhcp6OnTimerTick().

BOOLEAN Dhcp6LinkMovDetect (IN DHCP6_INSTANCE * *Instance*)

Detect Link movement for specified network device.

This routine will try to invoke Snp->GetStatus() to get the media status. If media present status switches from unpresent to present, a link movement is detected. Note that the underlying UNDI driver may not support reporting media status from GET_STATUS command. If that, fail to detect link movement.

Parameters:

[in] *Instance* The pointer to DHCP6_INSTANCE.

Return values:

TRUE A link movement is detected.

FALSE A link movement is not detected.

Referenced by Dhcp6OnTimerTick().

```
VOID EFIAPI Dhcp6OnTimerTick ( IN EFI_EVENT   Event,
                               IN VOID *      Context
                             )
```

The timer routine of the Dhcp6 instance for each second.

Parameters:

[in]	*Event*	The timer event.
[in]	*Context*	The opaque parameter to the function.

References _DHCP6_INSTANCE::AdPref, _DHCP6_INSTANCE::AdSelect, _DHCP6_INSTANCE::Config, Dhcp6CalculateExpireTime(), Dhcp6CleanupSession(), Dhcp6DequeueRetry(), Dhcp6IsValidTxCb(), Dhcp6LinkMovDetect(), Dhcp6MsgConfirm, Dhcp6MsgInfoRequest, Dhcp6MsgRenew, Dhcp6MsgSolicit, Dhcp6SelectAdvertiseMsg(), Dhcp6SendConfirmMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6TransmitPacket(), _DHCP6_TX_CB::Elapsed, _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, _DHCP6_IA_CB::LeaseTime, _DHCP6_TX_CB::RetryCnt, _DHCP6_TX_CB::RetryCtl, _DHCP6_TX_CB::RetryExp, _DHCP6_TX_CB::RetryLos, _DHCP6_TX_CB::SolicitRetry, _DHCP6_IA_CB::T1, _DHCP6_IA_CB::T2, _DHCP6_TX_CB::TickTime, _DHCP6_INSTANCE::TxList, _DHCP6_TX_CB::TxPacket, _DHCP6_INSTANCE::UdpSts, and _DHCP6_TX_CB::Xid.

Referenced by Dhcp6CreateInstance().

```
VOID EFIAPI Dhcp6ReceivePacket ( IN NET_BUF *            Udp6Wrap,
                                 IN UDP_END_POINT *      EndPoint,
                                 IN EFI_STATUS           IoStatus,
                                 IN VOID *               Context
                               )
```

The receive callback function for Dhcp6 exchange process.

Parameters:

[in]	*Udp6Wrap*	The pointer to the received net buffer.
[in]	*EndPoint*	The pointer to the udp end point.
[in]	*IoStatus*	The return status from udp io.
[in]	*Context*	The opaque parameter to the function.

References _DHCP6_SERVICE::Child, DHCP6_PACKET_ALL, Dhcp6CleanupRetry(), Dhcp6HandleStateful(), Dhcp6HandleStateless(), Dhcp6MsgInfoRequest, Dhcp6ReceivePacket(), _DHCP6_INSTANCE::TxList, _DHCP6_TX_CB::TxPacket, _DHCP6_SERVICE::UdpIo, and _DHCP6_TX_CB::Xid.

Referenced by Dhcp6HandleStateful(), Dhcp6HandleStateless(), Dhcp6ReceivePacket(), Dhcp6StartInfoRequest(), EfiDhcp6Decline(), EfiDhcp6Release(), EfiDhcp6RenewRebind(), and EfiDhcp6Start().

```
EFI_STATUS Dhcp6SeekStsOption ( IN DHCP6_INSTANCE *      Instance,
                                IN EFI_DHCP6_PACKET *    Packet,
                                OUT UINT8 **             Option
```

)

Seek StatusCode Option in package. A Status Code option may appear in the options field of a DHCP message and/or in the options field of another option. See details in section 22.13, RFC3315.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*Packet*	The pointer to reply messages.
[out]	*Option*	The pointer to status code option.

Return values:

EFI_SUCCESS	Seek status code option successfully.
EFI_DEVICE_ERROR	An unexpected error.

References Dhcp6OptIana, Dhcp6OptStatusCode, Dhcp6SeekIaOption(), Dhcp6SeekOption(), and Dhcp6StsSuccess.

Referenced by Dhcp6HandleAdvertiseMsg(), and Dhcp6HandleReplyMsg().

EFI_STATUS Dhcp6SelectAdvertiseMsg (IN DHCP6_INSTANCE * *Instance*,
IN EFI_DHCP6_PACKET * *AdSelect*
)

Select the appointed Dhcp6 advertisement message.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*AdSelect*	The pointer to the selected Dhcp6 advertisement message.

Return values:

EFI_SUCCESS	Selected the right advertisement message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
Others	Failed to select the advertise message.

References Dhcp6CallbackUser(), Dhcp6DequeueRetry(), Dhcp6OptServerUnicast, Dhcp6SeekOption(), Dhcp6SendRequestMsg(), and Dhcp6UpdateIaInfo().

Referenced by Dhcp6HandleAdvertiseMsg(), and Dhcp6OnTimerTick().

EFI_STATUS Dhcp6SendConfirmMsg (IN DHCP6_INSTANCE * *Instance*)

Create the Confirm message and send it.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.

Return values:

EFI_SUCCESS	Created and sent the confirm message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.
Others	Failed to send the confirm message.

References _DHCP6_SERVICE::ClientId, DHCP6_BASE_PACKET_SIZE, Dhcp6AppendETOption(),

Dhcp6AppendIaOption(), Dhcp6AppendOption(), Dhcp6CallbackUser(), Dhcp6EnqueueRetry(), Dhcp6MsgConfirm, Dhcp6OptClientId, Dhcp6TransmitPacket(), and _DHCP6_SERVICE::Xid.

Referenced by Dhcp6OnTimerTick().

```
EFI_STATUS Dhcp6SendDeclineMsg ( IN DHCP6_INSTANCE *   Instance,
                                 IN EFI_DHCP6_IA *     DecIa
                               )
```

Create the decline message and send it.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*DecIa*	The pointer to the decline Ia.

Return values:

EFI_SUCCESS	Created and sent the decline message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.
Others	Failed to send the decline message.

References _DHCP6_SERVICE::ClientId, DHCP6_BASE_PACKET_SIZE, Dhcp6AppendETOption(), Dhcp6AppendIaOption(), Dhcp6AppendOption(), Dhcp6CallbackUser(), Dhcp6EnqueueRetry(), Dhcp6MsgDecline, Dhcp6OptClientId, Dhcp6OptServerId, Dhcp6SeekOption(), Dhcp6TransmitPacket(), and _DHCP6_SERVICE::Xid.

Referenced by EfiDhcp6Decline().

```
EFI_STATUS Dhcp6SendInfoRequestMsg ( IN DHCP6_INSTANCE *                Instance,
                                     IN DHCP6_INF_CB *                  InfCb,
                                     IN BOOLEAN                         SendClientId,
                                     IN EFI_DHCP6_PACKET_OPTION *       OptionRequest,
                                     IN UINT32                          OptionCount,
                                     IN EFI_DHCP6_PACKET_OPTION *       OptionList[],
                                     IN EFI_DHCP6_RETRANSMISSION *      Retransmission
                                   )
```

Create the information request message and send it.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*InfCb*	The pointer to the information request control block.
[in]	*SendClientId*	If TRUE, the client identifier option will be included in information request message. Otherwise, the client identifier option will not be included.
[in]	*OptionRequest*	The pointer to the option request option.
[in]	*OptionCount*	The number options in the OptionList.
[in]	*OptionList*	The array pointers to the appended options.
[in]	*Retransmission*	The pointer to the retransmission control.

Return values:

EFI_SUCCESS	Created and sent the info-request message successfully.

EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
Others	Failed to send the info-request message.

References _DHCP6_SERVICE::ClientId, DHCP6_BASE_PACKET_SIZE, Dhcp6AppendETOption(), Dhcp6AppendOption(), Dhcp6EnqueueRetry(), Dhcp6MsgInfoRequest, Dhcp6OptClientId, Dhcp6TransmitPacket(), and _DHCP6_SERVICE::Xid.

Referenced by Dhcp6StartInfoRequest().

```
EFI_STATUS Dhcp6SendReleaseMsg ( IN DHCP6_INSTANCE *   Instance,
                                 IN EFI_DHCP6_IA *     RelIa
                               )
```

Create the release message and send it.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*RelIa*	The pointer to the release Ia.

Return values:

EFI_SUCCESS	Created and sent the release message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.
Others	Failed to send the release message.

References _DHCP6_SERVICE::ClientId, DHCP6_BASE_PACKET_SIZE, Dhcp6AppendETOption(), Dhcp6AppendIaOption(), Dhcp6AppendOption(), Dhcp6CallbackUser(), Dhcp6EnqueueRetry(), Dhcp6MsgRelease, Dhcp6OptClientId, Dhcp6OptServerId, Dhcp6SeekOption(), Dhcp6TransmitPacket(), and _DHCP6_SERVICE::Xid.

Referenced by EfiDhcp6Release(), and EfiDhcp6Stop().

```
EFI_STATUS Dhcp6SendRenewRebindMsg ( IN DHCP6_INSTANCE *   Instance,
                                     IN BOOLEAN            RebindRequest
                                   )
```

Create the renew/rebind message and send it.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*RebindRequest*	If TRUE, it is a Rebind type message. Otherwise, it is a Renew type message.

Return values:

EFI_SUCCESS	Created and sent the renew/rebind message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.
Others	Failed to send the renew/rebind message.

References _DHCP6_SERVICE::ClientId, DHCP6_BASE_PACKET_SIZE, Dhcp6AppendETOption(), Dhcp6AppendIaOption(), Dhcp6AppendOption(), Dhcp6CallbackUser(), Dhcp6EnqueueRetry(), Dhcp6MsgRebind, Dhcp6MsgRenew, Dhcp6OptClientId, Dhcp6OptServerId, Dhcp6SeekOption(), Dhcp6TransmitPacket(), and _DHCP6_SERVICE::Xid.

Referenced by Dhcp6OnTimerTick(), and EfiDhcp6RenewRebind().

EFI_STATUS Dhcp6SendRequestMsg (IN DHCP6_INSTANCE * *Instance*)

Create the request message and send it.

Parameters:

[in] *Instance* The pointer to the Dhcp6 instance.

Return values:

EFI_SUCCESS	Created and sent the request message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.
Others	Failed to send the request message.

References _DHCP6_SERVICE::ClientId, DHCP6_BASE_PACKET_SIZE, Dhcp6AppendETOption(), Dhcp6AppendIaOption(), Dhcp6AppendOption(), Dhcp6CallbackUser(), Dhcp6EnqueueRetry(), Dhcp6MsgRequest, Dhcp6OptClientId, Dhcp6OptServerId, Dhcp6SeekOption(), Dhcp6TransmitPacket(), and _DHCP6_SERVICE::Xid.

Referenced by Dhcp6HandleReplyMsg(), and Dhcp6SelectAdvertiseMsg().

EFI_STATUS Dhcp6SendSolicitMsg (IN DHCP6_INSTANCE * *Instance*)

Create the solicit message and send it.

Parameters:

[in] *Instance* The pointer to the Dhcp6 instance.

Return values:

EFI_SUCCESS	Created and sent the solicit message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
Others	Failed to send the solicit message.

References _DHCP6_SERVICE::ClientId, DHCP6_BASE_PACKET_SIZE, Dhcp6AppendETOption(), Dhcp6AppendIaOption(), Dhcp6AppendOption(), Dhcp6CallbackUser(), Dhcp6EnqueueRetry(), Dhcp6MsgSolicit, Dhcp6OptClientId, Dhcp6TransmitPacket(), and _DHCP6_SERVICE::Xid.

Referenced by Dhcp6InitSolicitMsg(), and EfiDhcp6Start().

```
EFI_STATUS Dhcp6StartInfoRequest ( IN DHCP6_INSTANCE *                Instance,
                                   IN BOOLEAN                         SendClientId,
                                   IN EFI_DHCP6_PACKET_OPTION *       OptionRequest,
                                   IN UINT32                          OptionCount,
                                   IN EFI_DHCP6_PACKET_OPTION *       OptionList[],
                                   IN EFI_DHCP6_RETRANSMISSION *      Retransmission,
                                   IN EFI_EVENT                       TimeoutEvent,
                                   IN EFI_DHCP6_INFO_CALLBACK         ReplyCallback,
                                   IN VOID *                          CallbackContext
                                 )
```

Start the information request process.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*SendClientId*	If TRUE, the client identifier option will be included in information request message. Otherwise, the client identifier option will not be included.
[in]	*OptionRequest*	The pointer to the option request option.
[in]	*OptionCount*	The number options in the OptionList.
[in]	*OptionList*	The array pointers to the appended options.
[in]	*Retransmission*	The pointer to the retransmission control.
[in]	*TimeoutEvent*	The event of timeout.
[in]	*ReplyCallback*	The callback function when the reply was received.
[in]	*CallbackContext*	The pointer to the parameter passed to the callback.

Return values:

EFI_SUCCESS	Start the info-request process successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_NO_MAPPING	No source address is available for use.
Others	Failed to start the info-request process.

References _DHCP6_INF_CB::CallbackContext, Dhcp6ReceivePacket(), Dhcp6SendInfoRequestMsg(), _DHCP6_INF_CB::Link, _DHCP6_INF_CB::ReplyCallback, _DHCP6_INF_CB::TimeoutEvent, and _DHCP6_SERVICE::UdpIo.

Referenced by EfiDhcp6InfoRequest().

```
EFI_STATUS Dhcp6TransmitPacket ( IN DHCP6_INSTANCE *      Instance,
                                 IN EFI_DHCP6_PACKET *    Packet,
                                 IN UINT16 *              Elapsed
                               )
```

Transmit Dhcp6 message by udpio.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*Packet*	The pointer to transmit message.
[in]	*Elapsed*	The pointer to the elapsed time value to fill in.

Return values:

EFI_SUCCESS	Successfully transmitted the packet.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
Others	Failed to transmit the packet.

References DHCP6_PORT_CLIENT, DHCP6_PORT_SERVER, Dhcp6DummyExtFree(), Dhcp6OnTransmitted(), mAllDhcpRelayAndServersAddress, SetElapsedTime(), and _DHCP6_SERVICE::UdpIo.

Referenced by Dhcp6OnTimerTick(), Dhcp6SendConfirmMsg(), Dhcp6SendDeclineMsg(), Dhcp6SendInfoRequestMsg(), Dhcp6SendReleaseMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6SendRequestMsg(), and Dhcp6SendSolicitMsg().

```
EFI_STATUS Dhcp6UpdateIaInfo ( IN OUT DHCP6_INSTANCE *    Instance,
                               IN EFI_DHCP6_PACKET *      Packet
                             )
```

Update Ia according to the new reply message.

Parameters:

[in,out]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*Packet*	The pointer to reply messages.

Return values:

EFI_SUCCESS	Updated the Ia information successfully.
EFI_DEVICE_ERROR	An unexpected error.

References Dhcp6GenerateIaCb(), Dhcp6OptIana, Dhcp6OptStatusCode, Dhcp6SeekIaOption(), Dhcp6SeekOption(), and Dhcp6StsSuccess.

Referenced by Dhcp6HandleReplyMsg(), and Dhcp6SelectAdvertiseMsg().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Dhcp6Dxe/Dhcp6Io.h File Reference

Functions

VOID Dhcp6CleanupRetry (IN DHCP6_INSTANCE *Instance, IN UINT32 Scope)
VOID Dhcp6CleanupSession (IN OUT DHCP6_INSTANCE *Instance, IN EFI_STATUS Status)
EFI_STATUS Dhcp6SendSolicitMsg (IN DHCP6_INSTANCE *Instance)
EFI_STATUS Dhcp6SendRequestMsg (IN DHCP6_INSTANCE *Instance)
EFI_STATUS Dhcp6SendRenewRebindMsg (IN DHCP6_INSTANCE *Instance, IN BOOLEAN RebindRequest)
EFI_STATUS Dhcp6SendDeclineMsg (IN DHCP6_INSTANCE *Instance, IN EFI_DHCP6_IA *DecIa)
EFI_STATUS Dhcp6SendReleaseMsg (IN DHCP6_INSTANCE *Instance, IN EFI_DHCP6_IA *RelIa)
EFI_STATUS Dhcp6StartInfoRequest (IN DHCP6_INSTANCE *Instance, IN BOOLEAN SendClientId, IN EFI_DHCP6_PACKET_OPTION *OptionRequest, IN UINT32 OptionCount, IN EFI_DHCP6_PACKET_OPTION *OptionList[], IN EFI_DHCP6_RETRANSMISSION *Retransmission, IN EFI_EVENT TimeoutEvent, IN EFI_DHCP6_INFO_CALLBACK ReplyCallback, IN VOID *CallbackContext)
EFI_STATUS Dhcp6SendInfoRequestMsg (IN DHCP6_INSTANCE *Instance, IN DHCP6_INF_CB *InfCb, IN BOOLEAN SendClientId, IN EFI_DHCP6_PACKET_OPTION *OptionRequest, IN UINT32 OptionCount, IN EFI_DHCP6_PACKET_OPTION *OptionList[], IN EFI_DHCP6_RETRANSMISSION *Retransmission)
VOID EFIAPI Dhcp6ReceivePacket (IN NET_BUF *Udp6Wrap, IN UDP_END_POINT *EndPoint, IN EFI_STATUS IoStatus, IN VOID *Context)
VOID EFIAPI Dhcp6OnTimerTick (IN EFI_EVENT Event, IN VOID *Context)

Detailed Description

Dhcp6 internal functions declaration.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
VOID Dhcp6CleanupRetry ( IN DHCP6_INSTANCE *   Instance,
                         IN UINT32             Scope
                       )
```

Clean up the specific nodes in the retry list.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*Scope*	The scope of cleanup nodes.

References DHCP6_PACKET_ALL, DHCP6_PACKET_STATEFUL, DHCP6_PACKET_STATELESS, Dhcp6MsgInfoRequest, _DHCP6_INF_CB::Link, _DHCP6_TX_CB::Link, _DHCP6_INF_CB::TimeoutEvent, and _DHCP6_TX_CB::TxPacket.

Referenced by Dhcp6CleanupSession(), Dhcp6DestroyInstance(), Dhcp6HandleStateless(), and Dhcp6ReceivePacket().

```
VOID Dhcp6CleanupSession ( IN OUT DHCP6_INSTANCE *   Instance,
                           IN EFI_STATUS             Status
                         )
```

Clean up the session of the instance stateful exchange.

Parameters:

[in,out]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*Status*	The return status from udp.

References DHCP6_PACKET_STATEFUL, and Dhcp6CleanupRetry().

Referenced by Dhcp6HandleStateful(), Dhcp6OnTimerTick(), and EfiDhcp6Stop().

```
VOID EFIAPI Dhcp6OnTimerTick ( IN EFI_EVENT   Event,
                               IN VOID *      Context
                             )
```

The timer routine of the Dhcp6 instance for each second.

Parameters:

[in]	*Event*	The timer event.
[in]	*Context*	The opaque parameter to the function.

References _DHCP6_INSTANCE::AdPref, _DHCP6_INSTANCE::AdSelect, _DHCP6_INSTANCE::Config, Dhcp6CalculateExpireTime(), Dhcp6CleanupSession(), Dhcp6DequeueRetry(), Dhcp6IsValidTxCb(), Dhcp6LinkMovDetect(), Dhcp6MsgConfirm, Dhcp6MsgInfoRequest, Dhcp6MsgRenew, Dhcp6MsgSolicit, Dhcp6SelectAdvertiseMsg(), Dhcp6SendConfirmMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6TransmitPacket(), _DHCP6_TX_CB::Elapsed, _DHCP6_IA_CB::Ia, _DHCP6_INSTANCE::IaCb, _DHCP6_IA_CB::LeaseTime, _DHCP6_TX_CB::RetryCnt, _DHCP6_TX_CB::RetryCtl, _DHCP6_TX_CB::RetryExp, _DHCP6_TX_CB::RetryLos, _DHCP6_TX_CB::SolicitRetry, _DHCP6_IA_CB::T1, _DHCP6_IA_CB::T2, _DHCP6_TX_CB::TickTime, _DHCP6_INSTANCE::TxList, _DHCP6_TX_CB::TxPacket, _DHCP6_INSTANCE::UdpSts, and _DHCP6_TX_CB::Xid.

Referenced by Dhcp6CreateInstance().

```
VOID EFIAPI Dhcp6ReceivePacket ( IN NET_BUF *          Udp6Wrap,
                                 IN UDP_END_POINT *    EndPoint,
                                 IN EFI_STATUS         IoStatus,
                                 IN VOID *             Context
                               )
```

The receive callback function for the Dhcp6 exchange process.

Parameters:

[in]	*Udp6Wrap*	The pointer to the received net buffer.
[in]	*EndPoint*	The pointer to the udp end point.
[in]	*IoStatus*	The return status from udp io.
[in]	*Context*	The opaque parameter to the function.

The receive callback function for Dhcp6 exchange process.

Parameters:

[in]	*Udp6Wrap*	The pointer to the received net buffer.
[in]	*EndPoint*	The pointer to the udp end point.
[in]	*IoStatus*	The return status from udp io.
[in]	*Context*	The opaque parameter to the function.

References _DHCP6_SERVICE::Child, DHCP6_PACKET_ALL, Dhcp6CleanupRetry(), Dhcp6HandleStateful(), Dhcp6HandleStateless(), Dhcp6MsgInfoRequest, Dhcp6ReceivePacket(), _DHCP6_INSTANCE::TxList, _DHCP6_TX_CB::TxPacket, _DHCP6_SERVICE::UdpIo, and _DHCP6_TX_CB::Xid.

Referenced by Dhcp6HandleStateful(), Dhcp6HandleStateless(), Dhcp6ReceivePacket(), Dhcp6StartInfoRequest(), EfiDhcp6Decline(), EfiDhcp6Release(), EfiDhcp6RenewRebind(), and EfiDhcp6Start().

```
EFI_STATUS Dhcp6SendDeclineMsg ( IN DHCP6_INSTANCE *   Instance,
                                 IN EFI_DHCP6_IA *     DecIa
                               )
```

Create the decline message and send it.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*DecIa*	The pointer to the decline Ia.

Return values:

EFI_SUCCESS	Create and send the decline message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.
Others	Failed to send the decline message.

Create the decline message and send it.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*DecIa*	The pointer to the decline Ia.

Return values:

EFI_SUCCESS	Created and sent the decline message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.
Others	Failed to send the decline message.

References _DHCP6_SERVICE::ClientId, DHCP6_BASE_PACKET_SIZE, Dhcp6AppendETOption(), Dhcp6AppendIaOption(), Dhcp6AppendOption(), Dhcp6CallbackUser(), Dhcp6EnqueueRetry(), Dhcp6MsgDecline, Dhcp6OptClientId, Dhcp6OptServerId, Dhcp6SeekOption(), Dhcp6TransmitPacket(), and _DHCP6_SERVICE::Xid.

Referenced by EfiDhcp6Decline().

```
EFI_STATUS Dhcp6SendInfoRequestMsg ( IN DHCP6_INSTANCE *                Instance,
                                     IN DHCP6_INF_CB *                  InfCb,
                                     IN BOOLEAN                         SendClientId,
                                     IN EFI_DHCP6_PACKET_OPTION *       OptionRequest,
                                     IN UINT32                          OptionCount,
                                     IN EFI_DHCP6_PACKET_OPTION *       OptionList[],
                                     IN EFI_DHCP6_RETRANSMISSION *      Retransmission
                                   )
```

Create the information request message and send it.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*InfCb*	The pointer to the information request control block.
[in]	*SendClientId*	If TRUE, the client identifier option will be included in information request message. Otherwise, the client identifier option will not be included.
[in]	*OptionRequest*	The pointer to the option request option.
[in]	*OptionCount*	The number options in the OptionList.
[in]	*OptionList*	The array pointers to the appended options.
[in]	*Retransmission*	The pointer to the retransmission control.

Return values:

EFI_SUCCESS	Create and send the info-request message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
Others	Failed to send the info-request message.

Create the information request message and send it.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*InfCb*	The pointer to the information request control block.
[in]	*SendClientId*	If TRUE, the client identifier option will be included in information request message. Otherwise, the client identifier option will not be included.
[in]	*OptionRequest*	The pointer to the option request option.
[in]	*OptionCount*	The number options in the OptionList.
[in]	*OptionList*	The array pointers to the appended options.
[in]	*Retransmission*	The pointer to the retransmission control.

Return values:

EFI_SUCCESS	Created and sent the info-request message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
Others	Failed to send the info-request message.

References _DHCP6_SERVICE::ClientId, DHCP6_BASE_PACKET_SIZE, Dhcp6AppendETOption(), Dhcp6AppendOption(), Dhcp6EnqueueRetry(), Dhcp6MsgInfoRequest, Dhcp6OptClientId, Dhcp6TransmitPacket(), and _DHCP6_SERVICE::Xid.

Referenced by Dhcp6StartInfoRequest().

```
EFI_STATUS Dhcp6SendReleaseMsg ( IN DHCP6_INSTANCE *   Instance,
                                 IN EFI_DHCP6_IA *     RelIa
                               )
```

Create the release message and send it.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*RelIa*	The pointer to the release Ia.

Return values:

EFI_SUCCESS	Create and send the release message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.
Others	Failed to send the release message.

Create the release message and send it.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*RelIa*	The pointer to the release Ia.

Return values:

EFI_SUCCESS	Created and sent the release message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.
Others	Failed to send the release message.

References _DHCP6_SERVICE::ClientId, DHCP6_BASE_PACKET_SIZE, Dhcp6AppendETOption(), Dhcp6AppendIaOption(), Dhcp6AppendOption(), Dhcp6CallbackUser(), Dhcp6EnqueueRetry(), Dhcp6MsgRelease, Dhcp6OptClientId, Dhcp6OptServerId, Dhcp6SeekOption(), Dhcp6TransmitPacket(), and _DHCP6_SERVICE::Xid.

Referenced by EfiDhcp6Release(), and EfiDhcp6Stop().

```
EFI_STATUS Dhcp6SendRenewRebindMsg ( IN DHCP6_INSTANCE *   Instance,
                                     IN BOOLEAN            RebindRequest
                                   )
```

Create the renew/rebind message and send it.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*RebindRequest*	If TRUE, it is a Rebind type message. Otherwise, it is a Renew type message.

Return values:

EFI_SUCCESS	Create and send the renew/rebind message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.
Others	Failed to send the renew/rebind message.

Create the renew/rebind message and send it.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*RebindRequest*	If TRUE, it is a Rebind type message. Otherwise, it is a Renew type message.

Return values:

EFI_SUCCESS	Created and sent the renew/rebind message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.
Others	Failed to send the renew/rebind message.

References _DHCP6_SERVICE::ClientId, DHCP6_BASE_PACKET_SIZE, Dhcp6AppendETOption(), Dhcp6AppendIaOption(), Dhcp6AppendOption(), Dhcp6CallbackUser(), Dhcp6EnqueueRetry(), Dhcp6MsgRebind, Dhcp6MsgRenew, Dhcp6OptClientId, Dhcp6OptServerId, Dhcp6SeekOption(), Dhcp6TransmitPacket(), and _DHCP6_SERVICE::Xid.

Referenced by Dhcp6OnTimerTick(), and EfiDhcp6RenewRebind().

EFI_STATUS Dhcp6SendRequestMsg (IN DHCP6_INSTANCE * *Instance*)

Create the request message and send it.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.

Return values:

EFI_SUCCESS	Create and send the request message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.
Others	Failed to send the request message.

Create the request message and send it.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.

Return values:

EFI_SUCCESS	Created and sent the request message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.
Others	Failed to send the request message.

References _DHCP6_SERVICE::ClientId, DHCP6_BASE_PACKET_SIZE, Dhcp6AppendETOption(), Dhcp6AppendIaOption(), Dhcp6AppendOption(), Dhcp6CallbackUser(), Dhcp6EnqueueRetry(), Dhcp6MsgRequest, Dhcp6OptClientId, Dhcp6OptServerId, Dhcp6SeekOption(), Dhcp6TransmitPacket(), and _DHCP6_SERVICE::Xid.

Referenced by Dhcp6HandleReplyMsg(), and Dhcp6SelectAdvertiseMsg().

EFI_STATUS Dhcp6SendSolicitMsg (IN DHCP6_INSTANCE * *Instance*)

Create the solicit message and send it.

Parameters:

[in] *Instance* The pointer to Dhcp6 instance.

Return values:

EFI_SUCCESS	Create and send the solicit message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
Others	Failed to send the solicit message.

Create the solicit message and send it.

Parameters:

[in] *Instance* The pointer to the Dhcp6 instance.

Return values:

EFI_SUCCESS	Created and sent the solicit message successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
Others	Failed to send the solicit message.

References _DHCP6_SERVICE::ClientId, DHCP6_BASE_PACKET_SIZE, Dhcp6AppendETOption(), Dhcp6AppendIaOption(), Dhcp6AppendOption(), Dhcp6CallbackUser(), Dhcp6EnqueueRetry(), Dhcp6MsgSolicit, Dhcp6OptClientId, Dhcp6TransmitPacket(), and _DHCP6_SERVICE::Xid.

Referenced by Dhcp6InitSolicitMsg(), and EfiDhcp6Start().

```
EFI_STATUS Dhcp6StartInfoRequest ( IN DHCP6_INSTANCE *                Instance,
                                   IN BOOLEAN                         SendClientId,
                                   IN EFI_DHCP6_PACKET_OPTION *       OptionRequest,
                                   IN UINT32                          OptionCount,
                                   IN EFI_DHCP6_PACKET_OPTION *       OptionList[],
                                   IN EFI_DHCP6_RETRANSMISSION *      Retransmission,
                                   IN EFI_EVENT                       TimeoutEvent,
                                   IN EFI_DHCP6_INFO_CALLBACK         ReplyCallback,
                                   IN VOID *                          CallbackContext
                                 )
```

Start the information request process.

Parameters:

[in]	*Instance*	The pointer to the Dhcp6 instance.
[in]	*SendClientId*	If TRUE, the client identifier option will be included in information request message. Otherwise, the client identifier option will not be included.
[in]	*OptionRequest*	The pointer to the option request option.
[in]	*OptionCount*	The number options in the OptionList.
[in]	*OptionList*	The array pointers to the appended options.
[in]	*Retransmission*	The pointer to the retransmission control.

[in]	*TimeoutEvent*	The event of timeout.
[in]	*ReplyCallback*	The callback function when the reply was received.
[in]	*CallbackContext*	The pointer to the parameter passed to the callback.

Return values:

EFI_SUCCESS	Start the info-request process successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_NO_MAPPING	No source address is available for use.
Others	Failed to start the info-request process.

References _DHCP6_INF_CB::CallbackContext, Dhcp6ReceivePacket(), Dhcp6SendInfoRequestMsg(), _DHCP6_INF_CB::Link, _DHCP6_INF_CB::ReplyCallback, _DHCP6_INF_CB::TimeoutEvent, and _DHCP6_SERVICE::UdpIo.

Referenced by EfiDhcp6InfoRequest().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Dhcp6Dxe/Dhcp6Utility.c File Reference

Functions

EFI_DHCP6_DUID * Dhcp6GenerateClientId (IN EFI_SIMPLE_NETWORK_MODE *Mode)
EFI_STATUS Dhcp6CopyConfigData (IN EFI_DHCP6_CONFIG_DATA *DstCfg, IN EFI_DHCP6_CONFIG_DATA *SorCfg)
VOID Dhcp6CleanupConfigData (IN OUT EFI_DHCP6_CONFIG_DATA *CfgData)
VOID Dhcp6CleanupModeData (IN OUT EFI_DHCP6_MODE_DATA *ModeData)
UINT32 Dhcp6CalculateExpireTime (IN UINT32 Base, IN BOOLEAN IsFirstRt, IN BOOLEAN NeedSigned)
VOID Dhcp6CalculateLeaseTime (IN DHCP6_IA_CB *IaCb)
EFI_STATUS Dhcp6CheckAddress (IN EFI_DHCP6_IA *Ia, IN UINT32 AddressCount, IN EFI_IPv6_ADDRESS *Addresses)
EFI_DHCP6_IA * Dhcp6DepriveAddress (IN EFI_DHCP6_IA *Ia, IN UINT32 AddressCount, IN EFI_IPv6_ADDRESS *Addresses)
VOID EFIAPI Dhcp6DummyExtFree (IN VOID *Arg)
VOID EFIAPI Dhcp6OnTransmitted (IN NET_BUF *Wrap, IN UDP_END_POINT *EndPoint, IN EFI_STATUS IoStatus, IN VOID *Context)
UINT8 * Dhcp6AppendOption (IN OUT UINT8 *Buf, IN UINT16 OptType, IN UINT16 OptLen, IN UINT8 *Data)
UINT8 * Dhcp6AppendIaAddrOption (IN OUT UINT8 *Buf, IN EFI_DHCP6_IA_ADDRESS *IaAddr, IN UINT32 MessageType)
UINT8 * Dhcp6AppendIaOption (IN OUT UINT8 *Buf, IN EFI_DHCP6_IA *Ia, IN UINT32 T1, IN UINT32 T2, IN UINT32 MessageType)
UINT8 * Dhcp6AppendETOption (IN OUT UINT8 *Buf, IN DHCP6_INSTANCE *Instance, OUT UINT16 **Elapsed)
VOID SetElapsedTime (IN UINT16 *Elapsed, IN DHCP6_INSTANCE *Instance)
UINT8 * Dhcp6SeekOption (IN UINT8 *Buf, IN UINT32 SeekLen, IN UINT16 OptType)
UINT8 * Dhcp6SeekIaOption (IN UINT8 *Buf, IN UINT32 SeekLen, IN EFI_DHCP6_IA_DESCRIPTOR *IaDesc)
BOOLEAN Dhcp6AddrIsInCurrentIa (IN EFI_DHCP6_IA_ADDRESS *IaAddr, IN EFI_DHCP6_IA *CurrentIa)
VOID Dhcp6ParseAddrOption (IN EFI_DHCP6_IA *CurrentIa, IN UINT8 *IaInnerOpt, IN UINT16 IaInnerLen, OUT UINT32 *AddrNum, IN OUT EFI_DHCP6_IA_ADDRESS *AddrBuf)
EFI_STATUS Dhcp6GenerateIaCb (IN DHCP6_INSTANCE *Instance, IN UINT8 *IaInnerOpt, IN UINT16 IaInnerLen, IN UINT32 T1, IN UINT32 T2)
EFI_STATUS Dhcp6CacheIa (IN DHCP6_INSTANCE *Instance)
VOID Dhcp6AppendCacheIa (IN DHCP6_INSTANCE *Instance)
EFI_STATUS Dhcp6GetMappingTimeOut (IN EFI_IP6_CONFIG_PROTOCOL *Ip6Cfg, OUT UINTN *TimeOut)

Detailed Description

Dhcp6 support functions implementation.

(C) Copyright 2015 Hewlett-Packard Development Company, L.P.
Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
BOOLEAN Dhcp6AddrIsInCurrentIa ( IN EFI_DHCP6_IA_ADDRESS *  IaAddr,
                                 IN EFI_DHCP6_IA *          CurrentIa
                               )
```

Check whether the incoming IPv6 address in IaAddr is one of the maintained addresses in the IA control blcok.

Parameters:

[in]	*IaAddr*	The pointer to the IA Address to be checked.
[in]	*CurrentIa*	The pointer to the IA in IA control block.

Return values:

TRUE	Yes, this Address is already in IA control block.
FALSE	No, this Address is NOT in IA control block.

Referenced by Dhcp6ParseAddrOption().

```
VOID Dhcp6AppendCacheIa ( IN DHCP6_INSTANCE *  Instance )
```

Append CacheIa to the currrent IA. Meanwhile, clear CacheIa.ValidLifetime to 0.

Parameters:

[in]	*Instance*	The pointer to DHCP6 instance.

Referenced by Dhcp6HandleReplyMsg().

```
UINT8* Dhcp6AppendETOption ( IN OUT UINT8 *          Buf,
                             IN DHCP6_INSTANCE *     Instance,
                             OUT UINT16 **           Elapsed
                           )
```

Append the appointed Elapsed time option to Buf, and move Buf to the end.

Parameters:

[in,out]	*Buf*	The pointer to the position to append.
[in]	*Instance*	The pointer to the Dhcp6 instance.
[out]	*Elapsed*	The pointer to the elapsed time value in the generated packet.

Returns:

Buf The position to append the next Ia option.

References Dhcp6OptElapsedTime.

Referenced by Dhcp6SendConfirmMsg(), Dhcp6SendDeclineMsg(), Dhcp6SendInfoRequestMsg(), Dhcp6SendReleaseMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6SendRequestMsg(), and Dhcp6SendSolicitMsg().

```
UINT8* Dhcp6AppendIaAddrOption ( IN OUT UINT8 *                Buf,
                                 IN EFI_DHCP6_IA_ADDRESS *     IaAddr,
                                 IN UINT32                     MessageType
                               )
```

Append the appointed IA Address option to Buf, and move Buf to the end.

Parameters:

[in,out]	*Buf*	The pointer to the position to append.
[in]	*IaAddr*	The pointer to the IA Address.
[in]	*MessageType*	Message type of DHCP6 package.

Returns:

Buf The position to append the next option.

References Dhcp6MsgConfirm, and Dhcp6OptIaAddr.

Referenced by Dhcp6AppendIaOption().

```
UINT8* Dhcp6AppendIaOption ( IN OUT UINT8 *       Buf,
                             IN EFI_DHCP6_IA *    Ia,
                             IN UINT32            T1,
                             IN UINT32            T2,
                             IN UINT32            MessageType
                           )
```

Append the appointed Ia option to Buf, and move Buf to the end.

Parameters:

[in,out]	*Buf*	The pointer to the position to append.
[in]	*Ia*	The pointer to the Ia.
[in]	*T1*	The time of T1.
[in]	*T2*	The time of T2.
[in]	*MessageType*	Message type of DHCP6 package.

Returns:

Buf The position to append the next Ia option.

References Dhcp6AppendIaAddrOption(), and Dhcp6OptIana.

Referenced by Dhcp6SendConfirmMsg(), Dhcp6SendDeclineMsg(), Dhcp6SendReleaseMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6SendRequestMsg(), and Dhcp6SendSolicitMsg().

```
UINT8* Dhcp6AppendOption ( IN OUT UINT8 *   Buf,
                           IN UINT16        OptType,
                           IN UINT16        OptLen,
```

IN UINT8 * *Data*
)

Append the option to Buf, and move Buf to the end.

Parameters:

[in,out]	*Buf*	The pointer to the buffer.
[in]	*OptType*	The option type.
[in]	*OptLen*	The length of option contents.
[in]	*Data*	The pointer to the option content.

Returns:

Buf The position to append the next option.

Referenced by Dhcp6SendConfirmMsg(), Dhcp6SendDeclineMsg(), Dhcp6SendInfoRequestMsg(), Dhcp6SendReleaseMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6SendRequestMsg(), and Dhcp6SendSolicitMsg().

EFI_STATUS Dhcp6CacheIa (IN DHCP6_INSTANCE * *Instance*)

Cache the current IA configuration information.

Parameters:

[in]	*Instance*	The pointer to DHCP6 Instance.

Return values:

EFI_SUCCESS	Cache the current IA successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.

Referenced by Dhcp6HandleReplyMsg().

UINT32 Dhcp6CalculateExpireTime (IN UINT32 *Base*,
IN BOOLEAN *IsFirstRt*,
IN BOOLEAN *NeedSigned*
)

Calculate the expire time by the algorithm defined in rfc.

Parameters:

[in]	*Base*	The base value of the time.
[in]	*IsFirstRt*	If TRUE, it is the first time to calculate expire time.
[in]	*NeedSigned*	If TRUE, the the signed factor is needed.

Returns:

Expire The calculated result for the new expire time.

References DHCP6_10_BIT_MASK.

Referenced by Dhcp6EnqueueRetry(), and Dhcp6OnTimerTick().

VOID Dhcp6CalculateLeaseTime (IN DHCP6_IA_CB * *IaCb*)

Calculate the lease time by the algorithm defined in rfc.

Parameters:

[in] *IaCb* The pointer to the Ia control block.

Referenced by Dhcp6GenerateIaCb().

EFI_STATUS Dhcp6CheckAddress (IN EFI_DHCP6_IA * *Ia*,
IN UINT32 *AddressCount*,
IN EFI_IPv6_ADDRESS * *Addresses*
)

Check whether the addresses are all included by the configured Ia.

Parameters:

[in] *Ia* The pointer to the Ia.
[in] *AddressCount* The number of addresses.
[in] *Addresses* The pointer to the addresses buffer.

Return values:

EFI_SUCCESS The addresses are all included by the configured IA.
EFI_NOT_FOUND The addresses are not included by the configured IA.

Referenced by EfiDhcp6Decline(), and EfiDhcp6Release().

VOID Dhcp6CleanupConfigData (IN OUT EFI_DHCP6_CONFIG_DATA * *CfgData*)

Clean up the configure data.

Parameters:

[in,out] *CfgData* The pointer to the configure data.

Referenced by Dhcp6DestroyInstance(), EfiDhcp6Configure(), and EfiDhcp6GetModeData().

VOID Dhcp6CleanupModeData (IN OUT EFI_DHCP6_MODE_DATA * *ModeData*)

Clean up the mode data.

Parameters:

[in,out] *ModeData* The pointer to the mode data.

Referenced by EfiDhcp6GetModeData().

EFI_STATUS Dhcp6CopyConfigData (IN EFI_DHCP6_CONFIG_DATA * *DstCfg*,
IN EFI_DHCP6_CONFIG_DATA * *SorCfg*
)

Copy the Dhcp6 configure data.

Parameters:

[in] *DstCfg* The pointer to the destination configure data.

[in] *SorCfg* The pointer to the source configure data.

Return values:

EFI_SUCCESS	Copy the content from SorCfg from DstCfg successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.

Referenced by EfiDhcp6Configure(), and EfiDhcp6GetModeData().

```
EFI_DHCP6_IA* Dhcp6DepriveAddress ( IN EFI_DHCP6_IA *        Ia,
                                    IN UINT32                AddressCount,
                                    IN EFI_IPv6_ADDRESS *    Addresses
                                  )
```

Deprive the addresses from current Ia, and generate another eliminated Ia.

Parameters:

[in]	*Ia*	The pointer to the Ia.
[in]	*AddressCount*	The number of addresses.
[in]	*Addresses*	The pointer to the addresses buffer.

Return values:

NULL	If it failed to generate the deprived Ia.
others	The pointer to the deprived Ia.

Referenced by EfiDhcp6Decline(), and EfiDhcp6Release().

```
VOID EFIAPI Dhcp6DummyExtFree ( IN VOID * Arg )
```

The dummy ext buffer free callback routine.

Parameters:

[in] *Arg* The pointer to the parameter.

Referenced by Dhcp6TransmitPacket().

```
EFI_DHCP6_DUID* Dhcp6GenerateClientId ( IN EFI_SIMPLE_NETWORK_MODE * Mode )
```

Generate client Duid in the format of Duid-llt.

Parameters:

[in] *Mode* The pointer to the mode of SNP.

Return values:

NULL	If it failed to generate a client Id.
others	The pointer to the new client id.

References Dhcp6DuidTypeLlt, and Dhcp6DuidTypeUuid.

Referenced by Dhcp6CreateService().

```
EFI_STATUS Dhcp6GenerateIaCb ( IN DHCP6_INSTANCE *   Instance,
                               IN UINT8 *            IaInnerOpt,
                               IN UINT16             IaInnerLen,
                               IN UINT32             T1,
                               IN UINT32             T2
                             )
```

Create a control blcok for the Ia according to the corresponding options.

Parameters:

[in]	*Instance*	The pointer to DHCP6 Instance.
[in]	*IaInnerOpt*	The pointer to the inner options in the Ia option.
[in]	*IaInnerLen*	The length of all the inner options in the Ia option.
[in]	*T1*	T1 time in the Ia option.
[in]	*T2*	T2 time in the Ia option.

Return values:

EFI_NOT_FOUND	No valid IA option is found.
EFI_SUCCESS	Create an IA control block successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected error.

References Dhcp6CalculateLeaseTime(), and Dhcp6ParseAddrOption().

Referenced by Dhcp6UpdateIaInfo().

```
EFI_STATUS Dhcp6GetMappingTimeOut ( IN EFI_IP6_CONFIG_PROTOCOL *   Ip6Cfg,
                                    OUT UINTN *                    TimeOut
                                  )
```

Calculate the Dhcp6 get mapping timeout by adding additinal delay to the IP6 DAD transmits count.

Parameters:

[in]	*Ip6Cfg*	The pointer to Ip6 config protocol.
[out]	*TimeOut*	The time out value in 100ns units.

Return values:

EFI_INVALID_PARAMETER	Input parameters are invalid.
EFI_SUCCESS	Calculate the time out value successfully.

References DHCP6_DAD_ADDITIONAL_DELAY.

Referenced by EfiDhcp6InfoRequest().

```
VOID EFIAPI Dhcp6OnTransmitted ( IN NET_BUF *          Wrap,
                                 IN UDP_END_POINT *    EndPoint,
                                 IN EFI_STATUS         IoStatus,
                                 IN VOID *             Context
                               )
```

The callback routine once message transmitted.

Parameters:

[in]	*Wrap*	The pointer to the received net buffer.
[in]	*EndPoint*	The pointer to the udp end point.
[in]	*IoStatus*	The return status from udp io.
[in]	*Context*	The opaque parameter to the function.

Referenced by Dhcp6TransmitPacket().

```
VOID Dhcp6ParseAddrOption ( IN EFI_DHCP6_IA *                    CurrentIa,
                            IN UINT8 *                           IaInnerOpt,
                            IN UINT16                            IaInnerLen,
                            OUT UINT32 *                         AddrNum,
                            IN OUT EFI_DHCP6_IA_ADDRESS *        AddrBuf
                          )
```

Parse the address option and update the address infomation.

Parameters:

[in]	*CurrentIa*	The pointer to the Ia Address in control blcok.
[in]	*IaInnerOpt*	The pointer to the buffer.
[in]	*IaInnerLen*	The length to parse.
[out]	*AddrNum*	The number of addresses.
[in,out]	*AddrBuf*	The pointer to the address buffer.

References Dhcp6AddrIsInCurrentIa(), and Dhcp6OptIaAddr.

Referenced by Dhcp6GenerateIaCb().

```
UINT8* Dhcp6SeekIaOption ( IN UINT8 *                        Buf,
                           IN UINT32                         SeekLen,
                           IN EFI_DHCP6_IA_DESCRIPTOR *      IaDesc
                         )
```

Seek the address of the first byte of the Ia option header.

Parameters:

[in]	*Buf*	The pointer to the buffer.
[in]	*SeekLen*	The length to seek.
[in]	*IaDesc*	The pointer to the Ia descriptor.

Return values:

NULL	If it failed to seek the Ia option.
others	The position to the Ia option.

Referenced by Dhcp6HandleReplyMsg(), Dhcp6SeekStsOption(), and Dhcp6UpdateIaInfo().

```
UINT8* Dhcp6SeekOption ( IN UINT8 *   Buf,
                         IN UINT32    SeekLen,
                         IN UINT16    OptType
```

)

Seek the address of the first byte of the option header.

Parameters:

[in]	*Buf*	The pointer to the buffer.
[in]	*SeekLen*	The length to seek.
[in]	*OptType*	The option type.

Return values:

NULL	If it failed to seek the option.
others	The position to the option.

Referenced by Dhcp6HandleAdvertiseMsg(), Dhcp6HandleReplyMsg(), Dhcp6HandleStateful(), Dhcp6HandleStateless(), Dhcp6SeekStsOption(), Dhcp6SelectAdvertiseMsg(), Dhcp6SendDeclineMsg(), Dhcp6SendReleaseMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6SendRequestMsg(), and Dhcp6UpdateIaInfo().

```
VOID SetElapsedTime ( IN UINT16 *              Elapsed,
                      IN DHCP6_INSTANCE *      Instance
                    )
```

Set the elapsed time based on the given instance and the pointer to the elapsed time option.

Parameters:

[in]	*Elapsed*	The pointer to the position to append.
[in]	*Instance*	The pointer to the Dhcp6 instance.

Referenced by Dhcp6TransmitPacket().

NetworkPkg/Dhcp6Dxe/Dhcp6Utility.h File Reference

Defines

"memItemRight" valign="bottom">DHCP6_10_BIT_MASK 0x3ff

#define DHCP6_DAD_ADDITIONAL_DELAY 30000

Functions

EFI_DHCP6_DUID * Dhcp6GenerateClientId (IN EFI_SIMPLE_NETWORK_MODE *Mode)

EFI_STATUS Dhcp6CopyConfigData (IN EFI_DHCP6_CONFIG_DATA *DstCfg, IN EFI_DHCP6_CONFIG_DATA *SorCfg)

VOID Dhcp6CleanupConfigData (IN OUT EFI_DHCP6_CONFIG_DATA *CfgData)

VOID Dhcp6CleanupModeData (IN OUT EFI_DHCP6_MODE_DATA *ModeData)

UINT32 Dhcp6CalculateExpireTime (IN UINT32 Base, I BOOLEAN IsFirstRt, IN BOOLEAN NeedSigne

VOID Dhcp6CalculateLeaseTime (IN DHCP6_IA_CB *IaCb)

EFI_STATUS Dhcp6CheckAddress (IN EFI_DHCP6_IA *Ia, I UINT32 AddressCount, IN EFI_IPv6_ADDRES *Addresses)

EFI_DHCP6_IA * Dhcp6DepriveAddress (IN EFI_DHCP6_IA *Ia, UINT32 AddressCount, IN EFI_IPv6_ADDRES *Addresses)

VOID EFIAPI Dhcp6DummyExtFree (IN VOID *Arg)

VOID EFIAPI Dhcp6OnTransmitted (IN NET_BUF *Wrap, IN UDP_END_POINT *EndPoint, IN EFI_STATUS IoStatus, IN VOID *Context)

UINT8 * Dhcp6AppendOption (IN OUT UINT8 *Buf, IN UINT16 OptType, IN UINT16 OptLen, IN UINT *Data)

UINT8 * Dhcp6AppendIaOption (IN OUT UINT8 *Buf, I EFI_DHCP6_IA *Ia, IN UINT32 T1, IN UINT32 T2, IN UINT32 MessageType)

UINT8 * Dhcp6AppendETOption (IN OUT UINT8 *Buf, DHCP6_INSTANCE *Instance, OUT UINT16 **Elapsed)

VOID SetElapsedTime (IN UINT16 *Elapsed, IN DHCP6_INSTANCE *Instance)

UINT8 * Dhcp6SeekOption (IN UINT8 *Buf, IN UINT32 SeekLen, IN UINT16 OptType)

UINT8 * Dhcp6SeekIaOption (IN UINT8 *Buf, IN UINT3 SeekLen, IN EFI_DHCP6_IA_DESCRIPTOR *IaDesc)

VOID Dhcp6ParseAddrOption (IN EFI_DHCP6_IA *CurrentIa, IN UINT8 *IaInnerOpt, IN UINT16

IaInnerLen, OUT UINT32 *AddrNum, IN OUT EFI_DHCP6_IA_ADDRESS *AddrBuf)
EFI_STATUS Dhcp6GenerateIaCb (IN DHCP6_INSTANCE *Instance, IN UINT8 *IaInnerOpt, IN UINT16 IaInnerLen, IN UINT32 T1, IN UINT32 T2)
EFI_STATUS Dhcp6CacheIa (IN DHCP6_INSTANCE *Instance)
VOID Dhcp6AppendCacheIa (IN DHCP6_INSTANCE *Instance)
EFI_STATUS Dhcp6GetMappingTimeOut (IN EFI_IP6_CONFIG_PROTOCOL *Ip6Cfg, OUT UINTN *TimeOut)

Detailed Description

Dhcp6 support functions declaration.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Dhcp6CalculateExpireTime().

#define DHCP6_DAD_ADDITIONAL_DELAY 30000000

Referenced by Dhcp6GetMappingTimeOut().

Function Documentation

VOID Dhcp6AppendCacheIa (IN DHCP6_INSTANCE * *Instance*)

Append CacheIa to the currrent IA. Meanwhile, clear CacheIa.ValidLifetime to 0.

Parameters:

[in] *Instance* The pointer to DHCP6 instance.

Referenced by Dhcp6HandleReplyMsg().

UINT8* Dhcp6AppendETOption (IN OUT UINT8 * *Buf*,
IN DHCP6_INSTANCE * *Instance*,
OUT UINT16 ** *Elapsed*

)

Append the appointed Elapsed time option to Buf, and move Buf to the end.

Parameters:

[in,out]	*Buf*	The pointer to the position to append.
[in]	*Instance*	The pointer to the Dhcp6 instance.
[out]	*Elapsed*	The pointer to the elapsed time value in the generated packet.

Returns:

Buf The position to append the next Ia option.

References Dhcp6OptElapsedTime.

Referenced by Dhcp6SendConfirmMsg(), Dhcp6SendDeclineMsg(), Dhcp6SendInfoRequestMsg(), Dhcp6SendReleaseMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6SendRequestMsg(), and Dhcp6SendSolicitMsg().

```
UINT8* Dhcp6AppendIaOption ( IN OUT UINT8 *        Buf,
                             IN EFI_DHCP6_IA *     Ia,
                             IN UINT32             T1,
                             IN UINT32             T2,
                             IN UINT32             MessageType
                           )
```

Append the Ia option to Buf, and move Buf to the end.

Parameters:

[in,out]	*Buf*	The pointer to the position to append.
[in]	*Ia*	The pointer to the Ia.
[in]	*T1*	The time of T1.
[in]	*T2*	The time of T2.
[in]	*MessageType*	Message type of DHCP6 package.

Returns:

Buf The position to append the next Ia option.

Append the appointed Ia option to Buf, and move Buf to the end.

Parameters:

[in,out]	*Buf*	The pointer to the position to append.
[in]	*Ia*	The pointer to the Ia.
[in]	*T1*	The time of T1.
[in]	*T2*	The time of T2.
[in]	*MessageType*	Message type of DHCP6 package.

Returns:

Buf The position to append the next Ia option.

References Dhcp6AppendIaAddrOption(), and Dhcp6OptIana.

Referenced by Dhcp6SendConfirmMsg(), Dhcp6SendDeclineMsg(), Dhcp6SendReleaseMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6SendRequestMsg(), and Dhcp6SendSolicitMsg().

```
UINT8* Dhcp6AppendOption ( IN OUT UINT8 *   Buf,
                           IN UINT16        OptType,
                           IN UINT16        OptLen,
                           IN UINT8 *       Data
                           )
```

Append the appointed option to the buf, and move the buf to the end.

Parameters:

[in,out]	*Buf*	The pointer to buffer.
[in]	*OptType*	The option type.
[in]	*OptLen*	The lenght of option content.s
[in]	*Data*	The pointer to the option content.

Returns:

Buf The position to append the next option.

Append the option to Buf, and move Buf to the end.

Parameters:

[in,out]	*Buf*	The pointer to the buffer.
[in]	*OptType*	The option type.
[in]	*OptLen*	The length of option contents.
[in]	*Data*	The pointer to the option content.

Returns:

Buf The position to append the next option.

Referenced by Dhcp6SendConfirmMsg(), Dhcp6SendDeclineMsg(), Dhcp6SendInfoRequestMsg(), Dhcp6SendReleaseMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6SendRequestMsg(), and Dhcp6SendSolicitMsg().

```
EFI_STATUS Dhcp6CacheIa ( IN DHCP6_INSTANCE *   Instance   )
```

Cache the current IA configuration information.

Parameters:

[in]	*Instance*	The pointer to DHCP6 Instance.

Return values:

EFI_SUCCESS	Cache the current IA successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.

Referenced by Dhcp6HandleReplyMsg().

```
UINT32 Dhcp6CalculateExpireTime ( IN UINT32     Base,
                                  IN BOOLEAN    IsFirstRt,
                                  IN BOOLEAN    NeedSigned
                                  )
```

Calculate the expire time by the algorithm defined in rfc.

Parameters:

[in]	*Base*	The base value of the time.
[in]	*IsFirstRt*	If TRUE, it is the first time to calculate expire time.
[in]	*NeedSigned*	If TRUE, the the signed factor is needed.

Returns:

Expire The calculated result for the new expire time.

References DHCP6_10_BIT_MASK.

Referenced by Dhcp6EnqueueRetry(), and Dhcp6OnTimerTick().

VOID Dhcp6CalculateLeaseTime (IN DHCP6_IA_CB * *IaCb*)

Calculate the lease time by the algorithm defined in rfc.

Parameters:

[in]	*IaCb*	The pointer to the Ia control block.

Referenced by Dhcp6GenerateIaCb().

```
EFI_STATUS Dhcp6CheckAddress ( IN EFI_DHCP6_IA *        Ia,
                               IN UINT32                AddressCount,
                               IN EFI_IPv6_ADDRESS *    Addresses
                             )
```

Check whether the addresses are all included by the configured Ia.

Parameters:

[in]	*Ia*	The pointer to the Ia.
[in]	*AddressCount*	The number of addresses.
[in]	*Addresses*	The pointer to the addresses buffer.

Return values:

EFI_SUCCESS	The addresses are all included by the configured IA.
EFI_NOT_FOUND	The addresses are not included by the configured IA.

Referenced by EfiDhcp6Decline(), and EfiDhcp6Release().

VOID Dhcp6CleanupConfigData (IN OUT EFI_DHCP6_CONFIG_DATA * *CfgData*)

Clean up the configure data.

Parameters:

[in,out]	*CfgData*	The pointer to the configure data.

Referenced by Dhcp6DestroyInstance(), EfiDhcp6Configure(), and EfiDhcp6GetModeData().

VOID Dhcp6CleanupModeData (IN OUT EFI_DHCP6_MODE_DATA * *ModeData*)

Clean up the mode data.

Parameters:

`[in,out]` *ModeData* The pointer to the mode data.

Referenced by EfiDhcp6GetModeData().

```
EFI_STATUS Dhcp6CopyConfigData ( IN EFI_DHCP6_CONFIG_DATA *  DstCfg,
                                 IN EFI_DHCP6_CONFIG_DATA *  SorCfg
                               )
```

Copy the Dhcp6 configure data.

Parameters:

`[in]` *DstCfg* The pointer to the destination configure data.
`[in]` *SorCfg* The pointer to the source configure data.

Return values:

EFI_SUCCESS Copy the content from SorCfg from DstCfg successfully.
EFI_OUT_OF_RESOURCES Required system resources could not be allocated.

Referenced by EfiDhcp6Configure(), and EfiDhcp6GetModeData().

```
EFI_DHCP6_IA* Dhcp6DepriveAddress ( IN EFI_DHCP6_IA *          Ia,
                                    IN UINT32                  AddressCount,
                                    IN EFI_IPv6_ADDRESS *      Addresses
                                  )
```

Deprive the addresses from current Ia, and generate another eliminated Ia.

Parameters:

`[in]` *Ia* The pointer to the Ia.
`[in]` *AddressCount* The number of addresses.
`[in]` *Addresses* The pointer to the addresses buffer.

Return values:

NULL If failed to generate the deprived Ia.
others The pointer to the deprived Ia.

Deprive the addresses from current Ia, and generate another eliminated Ia.

Parameters:

`[in]` *Ia* The pointer to the Ia.
`[in]` *AddressCount* The number of addresses.
`[in]` *Addresses* The pointer to the addresses buffer.

Return values:

NULL If it failed to generate the deprived Ia.
others The pointer to the deprived Ia.

Referenced by EfiDhcp6Decline(), and EfiDhcp6Release().

VOID EFIAPI Dhcp6DummyExtFree (IN VOID * *Arg*)
The dummy ext buffer free callback routine.

Parameters:
[in] *Arg* The pointer to the parameter.

Referenced by Dhcp6TransmitPacket().

EFI_DHCP6_DUID* Dhcp6GenerateClientId (IN EFI_SIMPLE_NETWORK_MODE * *Mode*)
Generate client Duid in the format of Duid-llt.

Parameters:
[in] *Mode* The pointer to the mode of SNP.

Return values:
NULL if failed to generate client Id.
Others The pointer to the new client id.

Generate client Duid in the format of Duid-llt.

Parameters:
[in] *Mode* The pointer to the mode of SNP.

Return values:
NULL If it failed to generate a client Id.
others The pointer to the new client id.

References Dhcp6DuidTypeLlt, and Dhcp6DuidTypeUuid.

Referenced by Dhcp6CreateService().

EFI_STATUS Dhcp6GenerateIaCb (IN DHCP6_INSTANCE * *Instance*,
IN UINT8 * *IaInnerOpt*,
IN UINT16 *IaInnerLen*,
IN UINT32 *T1*,
IN UINT32 *T2*
)
Create a control blcok for the Ia according to the corresponding options.

Parameters:
[in] *Instance* The pointer to DHCP6 Instance.
[in] *IaInnerOpt* The pointer to the inner options in the Ia option.
[in] *IaInnerLen* The length of all the inner options in the Ia option.
[in] *T1* T1 time in the Ia option.
[in] *T2* T2 time in the Ia option.

Return values:
EFI_NOT_FOUND No valid IA option is found.
EFI_SUCCESS Create an IA control block successfully.
EFI_OUT_OF_RESOURCES Required system resources could not be allocated.

EFI_DEVICE_ERROR An unexpected error.

References Dhcp6CalculateLeaseTime(), and Dhcp6ParseAddrOption().

Referenced by Dhcp6UpdateIaInfo().

```
EFI_STATUS Dhcp6GetMappingTimeOut ( IN EFI_IP6_CONFIG_PROTOCOL *  Ip6Cfg,
                                    OUT UINTN *                   TimeOut
                                  )
```

Calculate the Dhcp6 get mapping timeout by adding additinal delay to the IP6 DAD transmits count.

Parameters:

[in] *Ip6Cfg* The pointer to Ip6 config protocol.
[out] *TimeOut* The time out value in 100ns units.

Return values:

EFI_INVALID_PARAMETER Input parameters are invalid.
EFI_SUCCESS Calculate the time out value successfully.

References DHCP6_DAD_ADDITIONAL_DELAY.

Referenced by EfiDhcp6InfoRequest().

```
VOID EFIAPI Dhcp6OnTransmitted ( IN NET_BUF *          Wrap,
                                 IN UDP_END_POINT *    EndPoint,
                                 IN EFI_STATUS         IoStatus,
                                 IN VOID *             Context
                               )
```

The callback routine once message transmitted.

Parameters:

[in] *Wrap* The pointer to the received net buffer.
[in] *EndPoint* The pointer to the udp end point.
[in] *IoStatus* The return status from udp io.
[in] *Context* The opaque parameter to the function.

Referenced by Dhcp6TransmitPacket().

```
VOID Dhcp6ParseAddrOption ( IN EFI_DHCP6_IA *                CurrentIa,
                            IN UINT8 *                       IaInnerOpt,
                            IN UINT16                        IaInnerLen,
                            OUT UINT32 *                     AddrNum,
                            IN OUT EFI_DHCP6_IA_ADDRESS *    AddrBuf
                          )
```

Parse the address option and update the address info.

Parameters:

[in] *CurrentIa* The pointer to the Ia Address in control blcok.

[in]	*IaInnerOpt*	The pointer to the buffer.
[in]	*IaInnerLen*	The length to parse.
[out]	*AddrNum*	The number of addresses.
[in,out]	*AddrBuf*	The pointer to the address buffer.

Parse the address option and update the address infomation.

Parameters:

[in]	*CurrentIa*	The pointer to the Ia Address in control blcok.
[in]	*IaInnerOpt*	The pointer to the buffer.
[in]	*IaInnerLen*	The length to parse.
[out]	*AddrNum*	The number of addresses.
[in,out]	*AddrBuf*	The pointer to the address buffer.

References Dhcp6AddrIsInCurrentIa(), and Dhcp6OptIaAddr.

Referenced by Dhcp6GenerateIaCb().

```
UINT8* Dhcp6SeekIaOption ( IN UINT8 *                    Buf,
                           IN UINT32                     SeekLen,
                           IN EFI_DHCP6_IA_DESCRIPTOR *  IaDesc
                         )
```

Seek the address of the first byte of the Ia option header.

Parameters:

[in]	*Buf*	The pointer to the buffer.
[in]	*SeekLen*	The length to seek.
[in]	*IaDesc*	The pointer to the Ia descriptor.

Return values:

NULL	If failed to seek the Ia option.
others	The position to the Ia option.

Seek the address of the first byte of the Ia option header.

Parameters:

[in]	*Buf*	The pointer to the buffer.
[in]	*SeekLen*	The length to seek.
[in]	*IaDesc*	The pointer to the Ia descriptor.

Return values:

NULL	If it failed to seek the Ia option.
others	The position to the Ia option.

Referenced by Dhcp6HandleReplyMsg(), Dhcp6SeekStsOption(), and Dhcp6UpdateIaInfo().

```
UINT8* Dhcp6SeekOption ( IN UINT8 *  Buf,
                         IN UINT32   SeekLen,
                         IN UINT16   OptType
```

)

Seek the address of the first byte of the option header.

Parameters:

[in]	*Buf*	The pointer to buffer.
[in]	*SeekLen*	The length to seek.
[in]	*OptType*	The option type.

Return values:

NULL	If failed to seek the option.
others	The position to the option.

Seek the address of the first byte of the option header.

Parameters:

[in]	*Buf*	The pointer to the buffer.
[in]	*SeekLen*	The length to seek.
[in]	*OptType*	The option type.

Return values:

NULL	If it failed to seek the option.
others	The position to the option.

Referenced by Dhcp6HandleAdvertiseMsg(), Dhcp6HandleReplyMsg(), Dhcp6HandleStateful(), Dhcp6HandleStateless(), Dhcp6SeekStsOption(), Dhcp6SelectAdvertiseMsg(), Dhcp6SendDeclineMsg(), Dhcp6SendReleaseMsg(), Dhcp6SendRenewRebindMsg(), Dhcp6SendRequestMsg(), and Dhcp6UpdateIaInfo().

```
VOID SetElapsedTime ( IN UINT16 *              Elapsed,
                      IN DHCP6_INSTANCE *      Instance
                    )
```

Set the elapsed time based on the given instance and the pointer to the elapsed time option.

Parameters:

[in]	*Elapsed*	The pointer to the position to append.
[in]	*Instance*	The pointer to the Dhcp6 instance.

Referenced by Dhcp6TransmitPacket().

NetworkPkg/DnsDxe/ComponentName.c File Reference

Functions

EFI_STATUS EFIAPI DnsComponentNameGetDriverName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN CHAR8 *Language, OUT CHAR16 **DriverName)
EFI_STATUS EFIAPI DnsComponentNameGetControllerName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE ChildHandle, IN CHAR8 *Language, OUT CHAR16 **ControllerName)
EFI_STATUS UpdateDns4Name (EFI_DNS4_PROTOCOL *Dns4)
EFI_STATUS UpdateDns6Name (EFI_DNS6_PROTOCOL *Dns6)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED
EFI_COMPONENT_NAME_PROTOCOL gDnsComponentName
GLOBAL_REMOVE_IF_UNREFERENCED
EFI_COMPONENT_NAME2_PROTOCOL gDnsComponentName2
GLOBAL_REMOVE_IF_UNREFERENCED
EFI_UNICODE_STRING_TABLE mDnsDriverNameTable []
GLOBAL_REMOVE_IF_UNREFERENCED
EFI_UNICODE_STRING_TABLE * gDnsControllerNameTable = NULL

Detailed Description

Implementation of EFI_COMPONENT_NAME_PROTOCOL and EFI_COMPONENT_NAME2_PROTOCOL protocol.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI DnsComponentNameGetControllerName (IN EFI_COMPONENT_NAME_PROTOCOL * *This*,
IN EFI_HANDLE *ControllerHandle*,
IN EFI_HANDLE *ChildHandle*,

```
                    IN CHAR8 *                          Language,
                    OUT CHAR16 **                       ControllerName
                  )
```

Retrieves a Unicode string that is the user readable name of the controller that is being managed by an EFI Driver.

Parameters:

This	A pointer to the EFI_COMPONENT_NAME_PROTOCOL instance.
ControllerHandle	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
ChildHandle	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
Language	A pointer to a three character ISO 639-2 language identifier. This is the language of the controller name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer.
ControllerName	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language, from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gDnsComponentName, UpdateDns4Name(), and UpdateDns6Name().

```
EFI_STATUS EFIAPI
DnsComponentNameGetDriverName   (  IN
                                   EFI_COMPONENT_NAME_PROTOCOL *   This,
                                   IN CHAR8 *                      Language,
                                   OUT CHAR16 **                   DriverName
                                )
```

Retrieves a Unicode string that is the user-readable name of the EFI Driver.

Parameters:

This	A pointer to the EFI_COMPONENT_NAME_PROTOCOL instance.
Language	A pointer to a three-character ISO 639-2 language identifier. This is the language of the driver name that that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer.
DriverName	

A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by This in the language specified by Language.

Return values:

EFI_SUCCESS	The Unicode string for the Driver specified by This and the language specified by Language was returned in DriverName.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	DriverName is NULL.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gDnsComponentName, and mDnsDriverNameTable.

EFI_STATUS UpdateDns4Name (EFI_DNS4_PROTOCOL * *Dns4*)

Update the component name for the Dns4 child handle.

Parameters:

Dns4 A pointer to the EFI_DNS4_PROTOCOL.

Return values:

EFI_SUCCESS	Update the ControllerNameTable of this instance successfully.
EFI_INVALID_PARAMETER	The input parameter is invalid.

References gDnsComponentName, and gDnsComponentName2.

Referenced by DnsComponentNameGetControllerName().

EFI_STATUS UpdateDns6Name (EFI_DNS6_PROTOCOL * *Dns6*)

Update the component name for the Dns6 child handle.

Parameters:

Dns6 A pointer to the EFI_DNS6_PROTOCOL.

Return values:

EFI_SUCCESS	Update the ControllerNameTable of this instance successfully.
EFI_INVALID_PARAMETER	The input parameter is invalid.

References gDnsComponentName, and gDnsComponentName2.

Referenced by DnsComponentNameGetControllerName().

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gDnsComponentName

Initial value:

```
{
 DnsComponentNameGetDriverName,
 DnsComponentNameGetControllerName,
 "eng"
}
```

Component Name Protocol instance

Referenced by DnsComponentNameGetControllerName(), DnsComponentNameGetDriverName(), DnsDriverEntryPoint(), UpdateDns4Name(), and UpdateDns6Name().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gDnsComponentName2

Initial value:

```
{
 (EFI_COMPONENT_NAME2_GET_DRIVER_NAME)     DnsComponentNameGetDriverName,
 (EFI_COMPONENT_NAME2_GET_CONTROLLER_NAME) DnsComponentNameGetControllerName,
 "en"
}
```

Component Name 2 Protocol instance

Referenced by DnsDriverEntryPoint(), UpdateDns4Name(), and UpdateDns6Name().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE* gDnsControllerNameTable = NULL

Referenced by Dns4DriverBindingStop(), and Dns6DriverBindingStop().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mDnsDriverNameTable[]

Initial value:

```
{
 { "eng;en", (CHAR16 *)L"DNS Network Service Driver" },
 { NULL, NULL }
}
```

Table of driver names

Referenced by DnsComponentNameGetDriverName().

NetworkPkg/DnsDxe/DnsDhcp.c File Reference

Functions

VOID EFIAPI TimeoutToGetMap (IN EFI_EVENT Event, IN VOID *Context)
EFI_STATUS EFIAPI DnsStartIp4 (IN EFI_HANDLE Controller, IN EFI_HANDLE Image)
VOID DnsInitSeedPacket (OUT EFI_DHCP4_PACKET *Seed, IN EFI_IP4_CONFIG2_INTERFACE_INFO *InterfaceInfo)
VOID EFIAPI DhcpCommonNotify (IN EFI_EVENT Event, IN VOID *Context)
EFI_STATUS ParseDhcp4Ack (IN EFI_DHCP4_PROTOCOL *Dhcp4, IN EFI_DHCP4_PACKET *Packet, IN DNS4_SERVER_INFOR *DnsServerInfor)
EFI_STATUS EFIAPI ParseDhcp6Ack (IN EFI_DHCP6_PROTOCOL *This, IN VOID *Context, IN EFI_DHCP6_PACKET *Packet)
EFI_STATUS GetDns4ServerFromDhcp4 (IN DNS_INSTANCE *Instance, OUT UINT32 *DnsServerCount, OUT EFI_IPv4_ADDRESS **DnsServerList)
EFI_STATUS GetDns6ServerFromDhcp6 (IN EFI_HANDLE Image, IN EFI_HANDLE Controller, OUT UINT32 *DnsServerCount, OUT EFI_IPv6_ADDRESS **DnsServerList)

Detailed Description

Functions implementation related with DHCPv4/v6 for DNS driver.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
VOID EFIAPI DhcpCommonNotify ( IN EFI_EVENT    Event,
                               IN VOID *       Context
                             )
```

The common notify function.

Parameters:

[in] *Event* The event signaled.
[in] *Context* The context.

Referenced by GetDns4ServerFromDhcp4().

```
VOID DnsInitSeedPacket ( OUT EFI_DHCP4_PACKET *                  Seed,
                         IN EFI_IP4_CONFIG2_INTERFACE_INFO *     InterfaceInfo
```

)

This function initialize the DHCP4 message instance.

This function will pad each item of dhcp4 message packet.

Parameters:

Seed	Pointer to the message instance of the DHCP4 packet.
InterfaceInfo	Pointer to the EFI_IP4_CONFIG2_INTERFACE_INFO instance.

References DHCP4_MAGIC, DHCP4_OPCODE_REQUEST, and DHCP4_TAG_EOP.

Referenced by GetDns4ServerFromDhcp4().

```
EFI_STATUS EFIAPI DnsStartIp4 ( IN EFI_HANDLE Controller,
                                IN EFI_HANDLE Image
                              )
```

Create an IP child, use it to start the auto configuration, then destroy it.

Parameters:

[in] *Controller*	The controller which has the service installed.
[in] *Image*	The image handle used to open service.

Return values:

EFI_SUCCESS	The configuration is done.
Others	Other errors as indicated.

References TimeoutToGetMap().

Referenced by GetDns4ServerFromDhcp4().

```
EFI_STATUS GetDns4ServerFromDhcp4 ( IN DNS_INSTANCE *          Instance,
                                    OUT UINT32 *               DnsServerCount,
                                    OUT EFI_IPv4_ADDRESS **    DnsServerList
                                  )
```

Parse the DHCP ACK to get Dns4 server information.

Parameters:

Instance	The DNS instance.
DnsServerCount	Retrieved Dns4 server Ip count.
DnsServerList	Retrieved Dns4 server Ip list.

Return values:

EFI_SUCCESS	The Dns4 information is got from the DHCP ACK.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_NO_MEDIA	There was a media error.
Others	Other errors as indicated.

References DHCP4_MSG_INFORM, DHCP4_TAG_DNS_SERVER, DHCP4_TAG_PARA_LIST, DHCP4_TAG_TYPE, DhcpCommonNotify(), DNS_TIME_TO_GETMAP, DnsInitSeedPacket(), DnsStartIp4(), IP4_ETHER_PROTO, ParseDhcp4Ack(), DNS4_SERVER_INFOR::ServerCount, and DNS4_SERVER_INFOR::ServerList.

Referenced by Dns4Configure().

```
EFI_STATUS GetDns6ServerFromDhcp6 ( IN EFI_HANDLE              Image,
                                    IN EFI_HANDLE              Controller,
                                    OUT UINT32 *               DnsServerCount,
                                    OUT EFI_IPv6_ADDRESS **    DnsServerList
                                  )
```

Parse the DHCP ACK to get Dns6 server information.

Parameters:

Image	The handle of the driver image.
Controller	The handle of the controller.
DnsServerCount	Retrieved Dns6 server Ip count.
DnsServerList	Retrieved Dns6 server Ip list.

Return values:

EFI_SUCCESS	The Dns6 information is got from the DHCP ACK.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_NO_MEDIA	There was a media error.
Others	Other errors as indicated.

References DHCP6_TAG_DNS_REQUEST, DHCP6_TAG_DNS_SERVER, DNS_TIME_TO_GETMAP, ParseDhcp6Ack(), DNS6_SERVER_INFOR::ServerCount, and DNS6_SERVER_INFOR::ServerList.

Referenced by Dns6Configure().

```
EFI_STATUS ParseDhcp4Ack ( IN EFI_DHCP4_PROTOCOL *   Dhcp4,
                           IN EFI_DHCP4_PACKET *     Packet,
                           IN DNS4_SERVER_INFOR *    DnsServerInfor
                         )
```

Parse the ACK to get required information

Parameters:

Dhcp4	The DHCP4 protocol.
Packet	Packet waiting for parse.
DnsServerInfor	The required Dns4 server information.

Return values:

EFI_SUCCESS	The DNS information is got from the DHCP ACK.
EFI_NO_MAPPING	DHCP failed to acquire address and other information.
EFI_DEVICE_ERROR	Other errors as indicated.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.

References DHCP4_TAG_DNS_SERVER.

Referenced by GetDns4ServerFromDhcp4().

```
EFI_STATUS EFIAPI ParseDhcp6Ack ( IN EFI_DHCP6_PROTOCOL *  This,
                                  IN VOID *                Context,
                                  IN EFI_DHCP6_PACKET *    Packet
                                )
```

EFI_DHCP6_INFO_CALLBACK is provided by the consumer of the EFI DHCPv6 Protocol instance to intercept events that occurs in the DHCPv6 Information Request exchange process.

Parameters:

This	Pointer to the EFI_DHCP6_PROTOCOL instance that is used to configure this callback function.
Context	Pointer to the context that is initialized in the EFI_DHCP6_PROTOCOL.InfoRequest().
Packet	Pointer to Reply packet that has been received. The EFI DHCPv6 Protocol instance is responsible for freeing the buffer.

Return values:

EFI_SUCCESS	The DNS information is got from the DHCP ACK.
EFI_DEVICE_ERROR	Other errors as indicated.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.

References DHCP6_TAG_DNS_SERVER, DNS6_SERVER_INFOR::ServerCount, and DNS6_SERVER_INFOR::ServerList.

Referenced by GetDns6ServerFromDhcp6().

```
VOID EFIAPI TimeoutToGetMap ( IN EFI_EVENT  Event,
                              IN VOID *     Context
                            )
```

The callback function for the timer event used to get map.

Parameters:

[in] *Event*	The event this function is registered to.
[in] *Context*	The context registered to the event.

Referenced by DnsStartIp4().

NetworkPkg/DnsDxe/DnsDhcp.h File Reference

Data Structures

struct DNS4_SERVER_INFOR
struct DNS6_SERVER_INFOR

Defines

"memItemRight" valign="bottom">IP4_ETHER_PROTO 0x0800
#define DHCP4_OPCODE_REQUEST 1
#define DHCP4_MAGIC 0x63538263
#define DHCP4_TAG_EOP 255
#define DHCP4_TAG_TYPE 53
#define DHCP4_MSG_REQUEST 3
#define DHCP4_MSG_INFORM 8
#define DHCP4_TAG_PARA_LIST 55
#define DHCP4_TAG_DNS_SERVER 6
#define DHCP6_TAG_DNS_REQUEST 6
#define DHCP6_TAG_DNS_SERVER 23

Functions

EFI_STATUS ParseDhcp4Ack (IN EFI_DHCP4_PROTOCOL *Dhcp4, IN EFI_DHCP4_PACKET *Packet, IN DNS4_SERVER_INFOR *DnsServerInfor)
EFI_STATUS EFIAPI ParseDhcp6Ack (IN EFI_DHCP6_PROTOCOL *This, IN VOID *Context, IN EFI_DHCP6_PACKET *Packet)
EFI_STATUS GetDns4ServerFromDhcp4 (IN DNS_INSTANCE *Instance, OUT UINT32 *DnsServerCount, OUT EFI_IPv4_ADDRESS **DnsServerList)
EFI_STATUS GetDns6ServerFromDhcp6 (IN EFI_HANDLE Image, IN EFI_HANDLE Controller, OUT UINT32 *DnsServerCount, OUT EFI_IPv6_ADDRESS **DnsServerList)

Detailed Description

Functions implementation related with DHCPv4/v6 for DNS driver.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at

http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by DnsInitSeedPacket().

#define DHCP4_MSG_INFORM 8

Referenced by GetDns4ServerFromDhcp4().

#define DHCP4_MSG_REQUEST 3

#define DHCP4_OPCODE_REQUEST 1

Referenced by DnsInitSeedPacket().

#define DHCP4_TAG_DNS_SERVER 6

Referenced by GetDns4ServerFromDhcp4(), and ParseDhcp4Ack().

#define DHCP4_TAG_EOP 255

Referenced by DnsInitSeedPacket().

#define DHCP4_TAG_PARA_LIST 55

Referenced by GetDns4ServerFromDhcp4(), and IScsiDoDhcp().

#define DHCP4_TAG_TYPE 53

Referenced by GetDns4ServerFromDhcp4().

#define DHCP6_TAG_DNS_REQUEST 6

Referenced by GetDns6ServerFromDhcp6().

#define DHCP6_TAG_DNS_SERVER 23

Referenced by GetDns6ServerFromDhcp6(), and ParseDhcp6Ack().

#define IP4_ETHER_PROTO 0x0800

Referenced by GetDns4ServerFromDhcp4().

Function Documentation

```
EFI_STATUS GetDns4ServerFromDhcp4 ( IN DNS_INSTANCE *          Instance,
                                    OUT UINT32 *               DnsServerCount,
                                    OUT EFI_IPv4_ADDRESS **    DnsServerList
                                  )
```

Parse the DHCP ACK to get Dns4 server information.

Parameters:

Instance	The DNS instance.
DnsServerCount	Retrieved Dns4 server Ip count.
DnsServerList	Retrieved Dns4 server Ip list.

Return values:

EFI_SUCCESS	The Dns4 information is got from the DHCP ACK.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_NO_MEDIA	There was a media error.
Others	Other errors as indicated.

References DHCP4_MSG_INFORM, DHCP4_TAG_DNS_SERVER, DHCP4_TAG_PARA_LIST, DHCP4_TAG_TYPE, DhcpCommonNotify(), DNS_TIME_TO_GETMAP, DnsInitSeedPacket(), DnsStartIp4(), IP4_ETHER_PROTO, ParseDhcp4Ack(), DNS4_SERVER_INFOR::ServerCount, and DNS4_SERVER_INFOR::ServerList.

Referenced by Dns4Configure().

```
EFI_STATUS GetDns6ServerFromDhcp6 ( IN EFI_HANDLE              Image,
                                    IN EFI_HANDLE              Controller,
                                    OUT UINT32 *               DnsServerCount,
                                    OUT EFI_IPv6_ADDRESS **    DnsServerList
                                  )
```

Parse the DHCP ACK to get Dns6 server information.

Parameters:

Image	The handle of the driver image.
Controller	The handle of the controller.
DnsServerCount	Retrieved Dns6 server Ip count.
DnsServerList	Retrieved Dns6 server Ip list.

Return values:

EFI_SUCCESS	The Dns6 information is got from the DHCP ACK.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_NO_MEDIA	There was a media error.
Others	Other errors as indicated.

References DHCP6_TAG_DNS_REQUEST, DHCP6_TAG_DNS_SERVER, DNS_TIME_TO_GETMAP, ParseDhcp6Ack(), DNS6_SERVER_INFOR::ServerCount, and DNS6_SERVER_INFOR::ServerList.

Referenced by Dns6Configure().

```
EFI_STATUS ParseDhcp4Ack ( IN EFI_DHCP4_PROTOCOL *   Dhcp4,
                           IN EFI_DHCP4_PACKET *     Packet,
                           IN DNS4_SERVER_INFOR *    DnsServerInfor
                         )
```

Parse the ACK to get required information

Parameters:

Dhcp4	The DHCP4 protocol.
Packet	Packet waiting for parse.
DnsServerInfor	The required Dns4 server information.

Return values:

EFI_SUCCESS	The DNS information is got from the DHCP ACK.
EFI_NO_MAPPING	DHCP failed to acquire address and other information.
EFI_DEVICE_ERROR	Other errors as indicated.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.

References DHCP4_TAG_DNS_SERVER.

Referenced by GetDns4ServerFromDhcp4().

```
EFI_STATUS EFIAPI ParseDhcp6Ack ( IN EFI_DHCP6_PROTOCOL *   This,
                                  IN VOID *                 Context,
                                  IN EFI_DHCP6_PACKET *     Packet
                                )
```

EFI_DHCP6_INFO_CALLBACK is provided by the consumer of the EFI DHCPv6 Protocol instance to intercept events that occurs in the DHCPv6 Information Request exchange process.

Parameters:

This	Pointer to the EFI_DHCP6_PROTOCOL instance that is used to configure this callback function.
Context	Pointer to the context that is initialized in the EFI_DHCP6_PROTOCOL.InfoRequest().
Packet	Pointer to Reply packet that has been received. The EFI DHCPv6 Protocol instance is responsible for freeing the buffer.

Return values:

EFI_SUCCESS	The DNS information is got from the DHCP ACK.
EFI_DEVICE_ERROR	Other errors as indicated.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.

References DHCP6_TAG_DNS_SERVER, DNS6_SERVER_INFOR::ServerCount, and DNS6_SERVER_INFOR::ServerList.

Referenced by GetDns6ServerFromDhcp6().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/DnsDxe/DnsDriver.c File Reference

Functions

VOID DnsDestroyInstance (IN DNS_INSTANCE *Instance)
EFI_STATUS DnsCreateInstance (IN DNS_SERVICE *Service, OUT DNS_INSTANCE **Instance)
EFI_STATUS EFIAPI DnsDestroyChildEntryInHandleBuffer (IN LIST_ENTRY *Entry, IN VOID *Context)
EFI_STATUS EFIAPI DnsConfigNullUdp (IN UDP_IO *UdpIo, IN VOID *Context)
VOID DnsDestroyService (IN DNS_SERVICE *DnsSb)
EFI_STATUS DnsCreateService (IN EFI_HANDLE Controller, IN EFI_HANDLE Image, IN UINT8 IpVersion, OUT DNS_SERVICE **Service)
EFI_STATUS EFIAPI DnsUnload (IN EFI_HANDLE ImageHandle)
EFI_STATUS EFIAPI DnsDriverEntryPoint (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)
EFI_STATUS EFIAPI Dns4DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI Dns4DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI Dns4DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)
EFI_STATUS EFIAPI Dns6DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI Dns6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI Dns6DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)
EFI_STATUS EFIAPI Dns4ServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE *ChildHandle)
EFI_STATUS EFIAPI Dns4ServiceBindingDestroyChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE ChildHandle)
EFI_STATUS EFIAPI Dns6ServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE *ChildHandle)
EFI_STATUS EFIAPI Dns6ServiceBindingDestroyChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE

ChildHandle)

Variables

EFI_DRIVER_BINDING_PROTOCOL gDns4DriverBinding
EFI_DRIVER_BINDING_PROTOCOL gDns6DriverBinding
EFI_SERVICE_BINDING_PROTOCOL mDns4ServiceBinding
EFI_SERVICE_BINDING_PROTOCOL mDns6ServiceBinding
DNS_DRIVER_DATA * mDriverData = NULL

Detailed Description

The driver binding and service binding protocol for DnsDxe driver.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI Dns4DriverBindingStart (	IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*
)		

Starts a device controller or a bus controller.

The Start() function is designed to be invoked from the EFI boot service ConnectController(). As a result, much of the error checking on the parameters to Start() has been moved into this common boot service. It is legal to call Start() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE. 2. If RemainingDevicePath is not NULL, then it must be a pointer to a naturally aligned EFI_DEVICE_PATH_PROTOCOL. 3. Prior to calling Start(), the Supported() function for the driver specified by This must have been called with the same calling parameters, and Supported() must have returned EFI_SUCCESS.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to start. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For a bus driver, if this parameter is NULL, then handles for all the children of Controller are created by this driver. If this parameter is not NULL and the first Device Path Node is not the End of Device Path Node, then

only the handle for the child device specified by the first Device Path Node of RemainingDevicePath is created by this driver. If the first Device Path Node of RemainingDevicePath is the End of Device Path Node, no child handle is created by this driver.

Return values:

EFI_SUCCESS	The device was started.
EFI_DEVICE_ERROR	The device could not be started due to a device error.Currently not implemented.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
Others	The driver failded to start the device.

References DnsCreateService(), DnsDestroyService(), _DNS_SERVICE::ServiceBinding, and _DNS_SERVICE::Timer.

```
EFI_STATUS EFIAPI
Dns4DriverBindingStop (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     ControllerHandle,
  IN UINTN                          NumberOfChildren,
  IN EFI_HANDLE *                   ChildHandleBuffer
)
```

Stops a device controller or a bus controller.

The Stop() function is designed to be invoked from the EFI boot service DisconnectController(). As a result, much of the error checking on the parameters to Stop() has been moved into this common boot service. It is legal to call Stop() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE that was used on a previous call to this same driver's Start() function. 2. The first NumberOfChildren handles of ChildHandleBuffer must all be a valid EFI_HANDLE. In addition, all of these handles must have been created in this driver's Start() function, and the Start() function must have called OpenProtocol() on ControllerHandle with an Attribute of EFI_OPEN_PROTOCOL_BY_CHILD_CONTROLLER.

Parameters:

`[in]`	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
`[in]`	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
`[in]`	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
`[in]`	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer, _DNS_SERVICE::Dns4ChildrenList, DNS_SERVICE_FROM_THIS, DnsDestroyChildEntryInHandleBuffer(), DnsDestroyService(), gDnsControllerNameTable, DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren, and DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding.

EFI_STATUS EFIAPI Dns4DriverBindingSupported	(	IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
		IN EFI_HANDLE	*ControllerHandle*,
		IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*
	)		

Tests to see if this driver supports a given controller. If a child device is provided, it further tests to see if this driver supports creating a handle for the specified child device.

This function checks to see if the driver specified by This supports the device specified by ControllerHandle. Drivers will typically use the device path attached to ControllerHandle and/or the services from the bus I/O abstraction attached to ControllerHandle to determine if the driver supports ControllerHandle. This function may be called many times during platform initialization. In order to reduce boot times, the tests performed by this function must be very small, and take as little time as possible to execute. This function must not change the state of any hardware devices, and this function must be aware that the device specified by ControllerHandle may already be managed by the same driver or a different driver. This function must match its calls to AllocatePages() with FreePages(), AllocatePool() with FreePool(), and OpenProtocol() with CloseProtocol(). Because ControllerHandle may have been previously started by the same driver, if a protocol is already in the opened state, then it must not be closed with CloseProtocol(). This is required to guarantee the state of ControllerHandle is not modified by this function.

Parameters:

`[in]`	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
`[in]`	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
`[in]`	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by the driver specified by This.
EFI_ACCESS_DENIED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by a different driver or an application that requires exclusive access. Currently not implemented.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

EFI_STATUS EFIAPI Dns4ServiceBindingCreateChild	(	IN EFI_SERVICE_BINDING_PROTOCOL *	*This*,
		IN EFI_HANDLE *	*ChildHandle*
	)		

Creates a child handle and installs a protocol.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

[in]	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
[in]	*ChildHandle*	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.
EFI_INVALID_PARAMETER	ChildHandle is NULL.
EFI_OUT_OF_RESOURCES	There are not enough resources availabe to create the child
other	The child handle was not created

References _DNS_INSTANCE::ChildHandle, _DNS_SERVICE::ConnectUdp, _DNS_INSTANCE::Dns4, _DNS_SERVICE::Dns4ChildrenList, _DNS_SERVICE::Dns4ChildrenNum, DNS_SERVICE_FROM_THIS, DnsCreateInstance(), DnsDestroyInstance(), gDns4DriverBinding, _DNS_INSTANCE::Link, and _DNS_INSTANCE::UdpIo.

```
EFI_STATUS EFIAPI
Dns4ServiceBindingDestroyChild ( IN EFI_SERVICE_BINDING_PROTOCOL *  This,
                                 IN EFI_HANDLE                      ChildHandle
                               )
```

Destroys a child handle with a protocol installed on it.

The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

[in]	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
[in]	*ChildHandle*	Handle of the child to destroy

Return values:

EFI_SUCCES	The protocol was removed from ChildHandle.
EFI_UNSUPPORTED	ChildHandle does not support the protocol that is being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
EFI_ACCESS_DENIED	The protocol could not be removed from the ChildHandle because its services are being used.
other	The child handle was not destroyed

References _DNS_SERVICE::ConnectUdp, _DNS_SERVICE::Dns4ChildrenNum, DNS_INSTANCE_FROM_THIS_PROTOCOL4, DNS_SERVICE_FROM_THIS, DnsDestroyInstance(), gDns4DriverBinding, _DNS_INSTANCE::InDestroy, _DNS_INSTANCE::Link, _DNS_INSTANCE::Service, and _DNS_INSTANCE::UdpIo.

```
EFI_STATUS EFIAPI
Dns6DriverBindingStart ( IN EFI_DRIVER_BINDING_PROTOCOL *  This,
                         IN EFI_HANDLE                     ControllerHandle,
                         IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
                       )
```

Starts a device controller or a bus controller.

The Start() function is designed to be invoked from the EFI boot service ConnectController(). As a result, much of the error checking on the parameters to Start() has been moved into this common boot service. It is legal to call Start() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE. 2. If RemainingDevicePath is not NULL, then it must be a pointer to a naturally aligned EFI_DEVICE_PATH_PROTOCOL. 3. Prior to calling Start(), the Supported() function for the driver specified by This must have been called with the same calling parameters, and Supported() must have returned EFI_SUCCESS.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to start. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For a bus driver, if this parameter is NULL, then handles for all the children of Controller are created by this driver. If this parameter is not NULL and the first Device Path Node is not the End of Device Path Node, then only the handle for the child device specified by the first Device Path Node of RemainingDevicePath is created by this driver. If the first Device Path Node of RemainingDevicePath is the End of Device Path Node, no child handle is created by this driver.

Return values:

EFI_SUCCESS	The device was started.
EFI_DEVICE_ERROR	The device could not be started due to a device error.Currently not implemented.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
Others	The driver failded to start the device.

References DnsCreateService(), DnsDestroyService(), _DNS_SERVICE::ServiceBinding, and _DNS_SERVICE::Timer.

```
EFI_STATUS EFIAPI
Dns6DriverBindingStop
(
  IN EFI_DRIVER_BINDING_PROTOCOL *   This,
  IN EFI_HANDLE                      ControllerHandle,
  IN UINTN                           NumberOfChildren,
  IN EFI_HANDLE *                    ChildHandleBuffer
)
```

Stops a device controller or a bus controller.

The Stop() function is designed to be invoked from the EFI boot service DisconnectController(). As a result, much of the error checking on the parameters to Stop() has been moved into this common boot service. It is legal to call Stop() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE that was used on a previous call to this same driver's Start() function. 2. The first NumberOfChildren handles of ChildHandleBuffer must all be a valid EFI_HANDLE. In addition, all of these handles must have been created in this driver's Start() function, and the Start() function must have called OpenProtocol() on ControllerHandle with an Attribute of EFI_OPEN_PROTOCOL_BY_CHILD_CONTROLLER.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.

[in]	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
[in]	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
[in]	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer, _DNS_SERVICE::Dns6ChildrenList, DNS_SERVICE_FROM_THIS, DnsDestroyChildEntryInHandleBuffer(), DnsDestroyService(), gDnsControllerNameTable, DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren, and DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding.

```
EFI_STATUS EFIAPI
Dns6DriverBindingSupported (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
)
```

Tests to see if this driver supports a given controller. If a child device is provided, it further tests to see if this driver supports creating a handle for the specified child device.

This function checks to see if the driver specified by This supports the device specified by ControllerHandle. Drivers will typically use the device path attached to ControllerHandle and/or the services from the bus I/O abstraction attached to ControllerHandle to determine if the driver supports ControllerHandle. This function may be called many times during platform initialization. In order to reduce boot times, the tests performed by this function must be very small, and take as little time as possible to execute. This function must not change the state of any hardware devices, and this function must be aware that the device specified by ControllerHandle may already be managed by the same driver or a different driver. This function must match its calls to AllocatePages() with FreePages(), AllocatePool() with FreePool(), and OpenProtocol() with CloseProtocol(). Because ControllerHandle may have been previously started by the same driver, if a protocol is already in the opened state, then it must not be closed with CloseProtocol(). This is required to guarantee the state of ControllerHandle is not modified by this function.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by the driver specified by This.

EFI_ACCESS_DENIED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by a different driver or an application that requires exclusive access. Currently not implemented.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

```
EFI_STATUS EFIAPI
Dns6ServiceBindingCreateChild ( IN EFI_SERVICE_BINDING_PROTOCOL * This,
                                IN EFI_HANDLE * ChildHandle )
```

Creates a child handle and installs a protocol.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

[in]	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
[in]	*ChildHandle*	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.
EFI_INVALID_PARAMETER	ChildHandle is NULL.
EFI_OUT_OF_RESOURCES	There are not enough resources availabe to create the child
other	The child handle was not created

References _DNS_INSTANCE::ChildHandle, _DNS_SERVICE::ConnectUdp, _DNS_INSTANCE::Dns6, _DNS_SERVICE::Dns6ChildrenList, _DNS_SERVICE::Dns6ChildrenNum, DNS_SERVICE_FROM_THIS, DnsCreateInstance(), DnsDestroyInstance(), gDns6DriverBinding, _DNS_INSTANCE::Link, and _DNS_INSTANCE::UdpIo.

```
EFI_STATUS EFIAPI
Dns6ServiceBindingDestroyChild ( IN EFI_SERVICE_BINDING_PROTOCOL * This,
                                 IN EFI_HANDLE ChildHandle )
```

Destroys a child handle with a protocol installed on it.

The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

[in]	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
[in]	*ChildHandle*	Handle of the child to destroy

Return values:

EFI_SUCCES	The protocol was removed from ChildHandle.
EFI_UNSUPPORTED	ChildHandle does not support the protocol that is being removed.

EFI_INVALID_PARAMETER	Child handle is NULL.
EFI_ACCESS_DENIED	The protocol could not be removed from the ChildHandle because its services are being used.
other	The child handle was not destroyed

References _DNS_SERVICE::ConnectUdp, _DNS_SERVICE::Dns6ChildrenNum, DNS_INSTANCE_FROM_THIS_PROTOCOL6, DNS_SERVICE_FROM_THIS, DnsDestroyInstance(), gDns6DriverBinding, _DNS_INSTANCE::InDestroy, _DNS_INSTANCE::Link, _DNS_INSTANCE::Service, and _DNS_INSTANCE::UdpIo.

```
EFI_STATUS EFIAPI DnsConfigNullUdp ( IN UDP_IO * UdpIo,
                                     IN VOID * Context
                                   )
```

Config a NULL UDP that is used to keep the connection between UDP and DNS.

Just leave the Udp child unconfigured. When UDP is unloaded, DNS will be informed with DriverBinding Stop.

Parameters:

UdpIo	The UDP_IO to configure
Context	The opaque parameter to the callback

Return values:

EFI_SUCCESS	It always return EFI_SUCCESS directly.

Referenced by DnsCreateInstance(), and DnsCreateService().

```
EFI_STATUS DnsCreateInstance ( IN DNS_SERVICE * Service,
                               OUT DNS_INSTANCE ** Instance
                             )
```

Create the DNS instance and initialize it.

Parameters:

`[in]`	*Service*	The pointer to the DNS service.
`[out]`	*Instance*	The pointer to the DNS instance.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate resources.
EFI_SUCCESS	The DNS instance is created.

NicHandle

References _DNS_INSTANCE::Dns4, _DNS_INSTANCE::Dns4TxTokens, _DNS_INSTANCE::Dns6, _DNS_INSTANCE::Dns6TxTokens, DNS_INSTANCE_SIGNATURE, DNS_STATE_UNCONFIGED, DnsConfigNullUdp(), _DNS_INSTANCE::InDestroy, _DNS_INSTANCE::Link, mDns4Protocol,

mDns6Protocol, _DNS_INSTANCE::Service, _DNS_INSTANCE::Signature, _DNS_INSTANCE::State, and _DNS_INSTANCE::UdpIo.

Referenced by Dns4ServiceBindingCreateChild(), and Dns6ServiceBindingCreateChild().

```
EFI_STATUS DnsCreateService ( IN EFI_HANDLE        Controller,
                              IN EFI_HANDLE        Image,
                              IN UINT8             IpVersion,
                              OUT DNS_SERVICE **   Service
                            )
```

Create then initialize a Dns service binding instance.

Parameters:

Controller	The controller to install the DNS service binding on
Image	The driver binding image of the DNS driver
IpVersion	IpVersion for this service
Service	The variable to receive the created service binding instance.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate resource to create the instance.
EFI_DEVICE_ERROR	Failed to create a NULL UDP port to keep connection with UDP.
EFI_SUCCESS	The service instance is created for the controller.

References _DNS_SERVICE::ConnectUdp, _DNS_SERVICE::ControllerHandle, _DNS_SERVICE::Dns4ChildrenList, _DNS_SERVICE::Dns4ChildrenNum, _DNS_SERVICE::Dns6ChildrenList, _DNS_SERVICE::Dns6ChildrenNum, DNS_SERVICE_SIGNATURE, DnsConfigNullUdp(), DnsOnTimerRetransmit(), _DNS_SERVICE::ImageHandle, _DNS_SERVICE::IpVersion, mDns4ServiceBinding, mDns6ServiceBinding, _DNS_SERVICE::ServiceBinding, _DNS_SERVICE::Signature, _DNS_SERVICE::Timer, and _DNS_SERVICE::TimerToGetMap.

Referenced by Dns4DriverBindingStart(), and Dns6DriverBindingStart().

```
EFI_STATUS EFIAPI DnsDestroyChildEntryInHandleBuffer ( IN LIST_ENTRY *   Entry,
                                                       IN VOID *         Context
                                                     )
```

Callback function which provided by user to remove one node in NetDestroyLinkList process.

Parameters:

`[in]` *Entry*	The entry to be removed.
`[in]` *Context*	Pointer to the callback context corresponds to the Context in NetDestroyLinkList.

Return values:

EFI_SUCCESS	The entry has been removed successfully.
Others	Fail to remove the entry.

References _DNS_INSTANCE::ChildHandle, and DNS_INSTANCE_SIGNATURE.

Referenced by Dns4DriverBindingStop(), and Dns6DriverBindingStop().

VOID DnsDestroyInstance (IN DNS_INSTANCE * *Instance*)

Destroy the DNS instance and recycle the resources.

Parameters:

[in] *Instance* The pointer to the DNS instance.

References Dns4InstanceCancelToken(), and Dns6InstanceCancelToken().

Referenced by Dns4ServiceBindingCreateChild(), Dns4ServiceBindingDestroyChild(), Dns6ServiceBindingCreateChild(), and Dns6ServiceBindingDestroyChild().

VOID DnsDestroyService (IN DNS_SERVICE * *DnsSb*)

Release all the resource used the DNS service binding instance.

Parameters:

DnsSb The Dns service binding instance.

Referenced by Dns4DriverBindingStart(), Dns4DriverBindingStop(), Dns6DriverBindingStart(), and Dns6DriverBindingStop().

EFI_STATUS EFIAPI DnsDriverEntryPoint (IN EFI_HANDLE *ImageHandle*,
IN EFI_SYSTEM_TABLE * *SystemTable*
)

This is the declaration of an EFI image entry point. This entry point is the same for UEFI Applications, UEFI OS Loaders, and UEFI Drivers including both device drivers and bus drivers.

Parameters:

ImageHandle The firmware allocated handle for the UEFI image.
SystemTable A pointer to the EFI System Table.

Return values:

EFI_SUCCESS The operation completed successfully.
Others An unexpected error occurred.

References _DNS_DRIVER_DATA::Dns4CacheList, _DNS_DRIVER_DATA::Dns4ServerList, _DNS_DRIVER_DATA::Dns6CacheList, _DNS_DRIVER_DATA::Dns6ServerList, DnsOnTimerUpdate(), gDns4DriverBinding, gDns6DriverBinding, gDnsComponentName, gDnsComponentName2, and _DNS_DRIVER_DATA::Timer.

EFI_STATUS EFIAPI DnsUnload (IN EFI_HANDLE *ImageHandle*)

Unloads an image.

Parameters:

ImageHandle Handle that identifies the image to be unloaded.

Return values:

EFI_SUCCESS	The image has been unloaded.
EFI_INVALID_PARAMETER	ImageHandle is not a valid image handle.

References _DNS_DRIVER_DATA::Dns4CacheList, _DNS_DRIVER_DATA::Dns4ServerList, _DNS_DRIVER_DATA::Dns6CacheList, _DNS_DRIVER_DATA::Dns6ServerList, DNS6_CACHE::DnsCache, DNS4_CACHE::DnsCache, and _DNS_DRIVER_DATA::Timer.

Variable Documentation

EFI_DRIVER_BINDING_PROTOCOL gDns4DriverBinding

Initial value:

```
{
 Dns4DriverBindingSupported,
 Dns4DriverBindingStart,
 Dns4DriverBindingStop,
 DNS_VERSION,
 NULL,
 NULL
}
```

Referenced by Dns4ServiceBindingCreateChild(), Dns4ServiceBindingDestroyChild(), and DnsDriverEntryPoint().

EFI_DRIVER_BINDING_PROTOCOL gDns6DriverBinding

Initial value:

```
{
 Dns6DriverBindingSupported,
 Dns6DriverBindingStart,
 Dns6DriverBindingStop,
 DNS_VERSION,
 NULL,
 NULL
}
```

Referenced by Dns6ServiceBindingCreateChild(), Dns6ServiceBindingDestroyChild(), and DnsDriverEntryPoint().

EFI_SERVICE_BINDING_PROTOCOL mDns4ServiceBinding

Initial value:

```
{
 Dns4ServiceBindingCreateChild,
 Dns4ServiceBindingDestroyChild
}
```

Referenced by DnsCreateService().

EFI_SERVICE_BINDING_PROTOCOL mDns6ServiceBinding

Initial value:

```
{
 Dns6ServiceBindingCreateChild,
 Dns6ServiceBindingDestroyChild
}
```

Referenced by DnsCreateService().

DNS_DRIVER_DATA* mDriverData = NULL

Referenced by Dns4Configure(), Dns4GetModeData(), Dns4HostNameToIp(), Dns4UpdateDnsCache(), Dns6Configure(), Dns6GetModeData(), Dns6HostNameToIp(), Dns6UpdateDnsCache(), DnsOnTimerUpdate(), and ParseDnsResponse().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/DnsDxe/DnsDriver.h File Reference

Data Structures

struct _DNS_DRIV
struct _DNS_SERV
struct _DNS_INST
struct DNS_DESTI

Defines

"memItemRight" valign="bottom">DNS_SERVICE_SIGNATURE SIGNATURE_32 ('D', 'N', 'S', 'S')
#define DNS_INSTA
'N', 'S', 'I')
#define DNS_SERVI
ServiceBindi
#define DNS_INSTA
DNS_INSTA
#define DNS_INSTA
DNS_INSTA

Typedefs

typedef struct _DNS_DRIVER_DATA DNS_DRIVE
typedef struct _DNS_SERVICE DNS_SERVI
typedef struct _DNS_INSTANCE DNS_INSTA

Functions

VOID DnsDestroyI
EFI_STATUS DnsCreateIns
DNS_INSTA
EFI_STATUS EFIAPI DnsDestroyC
*Entry, IN V
EFI_STATUS EFIAPI DnsConfigNu
*Context)
VOID DnsDestroyS
EFI_STATUS DnsCreateSe
EFI_HANDI
DNS_SERVI
EFI_STATUS EFIAPI DnsUnload (
EFI_STATUS EFIAPI DnsDriverEn
EFI_SYSTE
EFI_STATUS EFIAPI Dns4DriverB
EFI_DRIVE
EFI_HANDI
EFI_DEVICI
*RemainingD
EFI_STATUS EFIAPI Dns4DriverB
EFI_DRIVE

EFI_HANDLE Contr
EFI_DEVICE_PATH
*RemainingDevicePa
EFI_STATUS EFIAPI Dns4DriverBindingS
EFI_DRIVER_BIND
EFI_HANDLE Contr
NumberOfChildren, I
*ChildHandleBuffer)
EFI_STATUS EFIAPI Dns6DriverBindingS
EFI_DRIVER_BIND
EFI_HANDLE Contr
EFI_DEVICE_PATH
*RemainingDevicePa
EFI_STATUS EFIAPI Dns6DriverBindingS
EFI_DRIVER_BIND
EFI_HANDLE Contr
EFI_DEVICE_PATH
*RemainingDevicePa
EFI_STATUS EFIAPI Dns6DriverBindingS
EFI_DRIVER_BIND
EFI_HANDLE Contr
NumberOfChildren, I
*ChildHandleBuffer)
EFI_STATUS EFIAPI Dns4ServiceBinding
EFI_SERVICE_BIN
EFI_HANDLE *Chil
EFI_STATUS EFIAPI Dns4ServiceBinding
EFI_SERVICE_BIN
EFI_HANDLE Child
EFI_STATUS EFIAPI Dns6ServiceBinding
EFI_SERVICE_BIN
EFI_HANDLE *Chil
EFI_STATUS EFIAPI Dns6ServiceBinding
EFI_SERVICE_BIN
EFI_HANDLE Child

Variables

DNS_DRIVER_DATA * mDriverData

Detailed Description

The header files of the driver binding and service binding protocol for DnsDxe driver.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"paramtype">a) CR (a, DNS_INSTANCE, Dns4, DNS_INSTANCE_SIGNATURE)

Referenced by Dns4Cancel(), Dns4Configure(), Dns4GetModeData(), Dns4HostNameToIp(), Dns4Poll(), and Dns4ServiceBindingDestroyChild().

#define DNS_INSTANCE_FROM_THIS_PROTOCOL6 (a) CR (a, DNS_INSTANCE, Dns6, DNS_INSTANCE_SIGNATURE)

Referenced by Dns6Cancel(), Dns6Configure(), Dns6GetModeData(), Dns6HostNameToIp(), Dns6Poll(), and Dns6ServiceBindingDestroyChild().

#define DNS_INSTANCE_SIGNATURE SIGNATURE_32 ('D', 'N', 'S', 'I')

Referenced by DnsCreateInstance(), DnsDestroyChildEntryInHandleBuffer(), DnsOnPacketReceived(), and DnsOnPacketSent().

#define DNS_SERVICE_FROM_THIS (a) CR (a, DNS_SERVICE, ServiceBinding, DNS_SERVICE_SIGNATURE)

Referenced by Dns4DriverBindingStop(), Dns4ServiceBindingCreateChild(), Dns4ServiceBindingDestroyChild(), Dns6DriverBindingStop(), Dns6ServiceBindingCreateChild(), and Dns6ServiceBindingDestroyChild().

#define DNS_SERVICE_SIGNATURE SIGNATURE_32 ('D', 'N', 'S', 'S')

Referenced by DnsCreateService().

Typedef Documentation

typedef struct _DNS_DRIVER_DATA DNS_DRIVER_DATA
Dns service block

typedef struct _DNS_INSTANCE DNS_INSTANCE
Dns instance block

typedef struct _DNS_SERVICE DNS_SERVICE
Dns service block

Function Documentation

EFI_STATUS EFIAPI Dns4DriverBindingStart	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*
	)	

Starts a device controller or a bus controller.

The Start() function is designed to be invoked from the EFI boot service ConnectController(). As a result, much of the error checking on the parameters to Start() has been moved into this common boot service. It is legal to call Start() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE. 2. If RemainingDevicePath is not NULL, then it must be a pointer to a naturally aligned EFI_DEVICE_PATH_PROTOCOL. 3. Prior to calling Start(), the Supported() function for the driver specified by This must have been called with the same calling parameters, and Supported() must have returned EFI_SUCCESS.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to start. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For a bus driver, if this parameter is NULL, then handles for all the children of Controller are created by this driver. If this parameter is not NULL and the first Device Path Node is not the End of Device Path Node, then only the handle for the child device specified by the first Device Path Node of RemainingDevicePath is created by this driver. If the first Device Path Node of RemainingDevicePath is the End of Device Path Node, no child handle is created by this driver.

Return values:

EFI_SUCCESS	The device was started.
EFI_DEVICE_ERROR	The device could not be started due to a device error.Currently not implemented.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
Others	The driver failded to start the device.

References DnsCreateService(), DnsDestroyService(), _DNS_SERVICE::ServiceBinding, and _DNS_SERVICE::Timer.

EFI_STATUS EFIAPI Dns4DriverBindingStop	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN UINTN	*NumberOfChildren*,
	IN EFI_HANDLE *	*ChildHandleBuffer*
	)	

Stops a device controller or a bus controller.

The Stop() function is designed to be invoked from the EFI boot service DisconnectController(). As a result, much of the error checking on the parameters to Stop() has been moved into this common boot service. It is legal to call Stop() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE that was used on a previous call to this same driver's Start() function. 2. The first NumberOfChildren handles of ChildHandleBuffer must all be a valid EFI_HANDLE. In addition, all of these handles must have been created in this driver's Start() function, and the Start() function must have called OpenProtocol() on ControllerHandle with an Attribute of EFI_OPEN_PROTOCOL_BY_CHILD_CONTROLLER.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
[in]	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
[in]	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer, _DNS_SERVICE::Dns4ChildrenList, DNS_SERVICE_FROM_THIS, DnsDestroyChildEntryInHandleBuffer(), DnsDestroyService(), gDnsControllerNameTable, DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren, and DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding.

```
EFI_STATUS EFIAPI
Dns4DriverBindingSupported (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
)
```

Tests to see if this driver supports a given controller. If a child device is provided, it further tests to see if this driver supports creating a handle for the specified child device.

This function checks to see if the driver specified by This supports the device specified by ControllerHandle. Drivers will typically use the device path attached to ControllerHandle and/or the services from the bus I/O abstraction attached to ControllerHandle to determine if the driver supports ControllerHandle. This function may be called many times during platform initialization. In order to reduce boot times, the tests performed by this function must be very small, and take as little time as possible to execute. This function must not change the state of any hardware devices, and this function must be aware that the device specified by ControllerHandle may already be managed by the same driver or a different driver. This function must match its calls to AllocatePages() with FreePages(), AllocatePool() with FreePool(), and OpenProtocol() with CloseProtocol(). Because ControllerHandle may have been previously started by the same driver, if a protocol is already in the opened state, then it must not be closed with CloseProtocol(). This is required to guarantee the state of ControllerHandle is not modified by this function.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	

A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by the driver specified by This.
EFI_ACCESS_DENIED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by a different driver or an application that requires exclusive access. Currently not implemented.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

```
EFI_STATUS EFIAPI
Dns4ServiceBindingCreateChild ( IN EFI_SERVICE_BINDING_PROTOCOL *  This,
                                IN EFI_HANDLE *                   ChildHandle
                              )
```

Creates a child handle and installs a protocol.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

`[in]` *This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
`[in]` *ChildHandle*	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.
EFI_INVALID_PARAMETER	ChildHandle is NULL.
EFI_OUT_OF_RESOURCES	There are not enough resources availabe to create the child
other	The child handle was not created

References _DNS_INSTANCE::ChildHandle, _DNS_SERVICE::ConnectUdp, _DNS_INSTANCE::Dns4, _DNS_SERVICE::Dns4ChildrenList, _DNS_SERVICE::Dns4ChildrenNum, DNS_SERVICE_FROM_THIS, DnsCreateInstance(), DnsDestroyInstance(), gDns4DriverBinding, _DNS_INSTANCE::Link, and _DNS_INSTANCE::UdpIo.

```
EFI_STATUS EFIAPI
Dns4ServiceBindingDestroyChild ( IN EFI_SERVICE_BINDING_PROTOCOL *  This,
                                 IN EFI_HANDLE                      ChildHandle
                               )
```

Destroys a child handle with a protocol installed on it.

The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

[in]	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
[in]	*ChildHandle*	Handle of the child to destroy

Return values:

EFI_SUCCES	The protocol was removed from ChildHandle.
EFI_UNSUPPORTED	ChildHandle does not support the protocol that is being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
EFI_ACCESS_DENIED	The protocol could not be removed from the ChildHandle because its services are being used.
other	The child handle was not destroyed

References _DNS_SERVICE::ConnectUdp, _DNS_SERVICE::Dns4ChildrenNum, DNS_INSTANCE_FROM_THIS_PROTOCOL4, DNS_SERVICE_FROM_THIS, DnsDestroyInstance(), gDns4DriverBinding, _DNS_INSTANCE::InDestroy, _DNS_INSTANCE::Link, _DNS_INSTANCE::Service, and _DNS_INSTANCE::UdpIo.

```
EFI_STATUS EFIAPI
Dns6DriverBindingStart ( IN EFI_DRIVER_BINDING_PROTOCOL *  This,
                         IN EFI_HANDLE                     ControllerHandle,
                         IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
                       )
```

Starts a device controller or a bus controller.

The Start() function is designed to be invoked from the EFI boot service ConnectController(). As a result, much of the error checking on the parameters to Start() has been moved into this common boot service. It is legal to call Start() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE. 2. If RemainingDevicePath is not NULL, then it must be a pointer to a naturally aligned EFI_DEVICE_PATH_PROTOCOL. 3. Prior to calling Start(), the Supported() function for the driver specified by This must have been called with the same calling parameters, and Supported() must have returned EFI_SUCCESS.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to start. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For a bus driver, if this parameter is NULL, then handles for all the children of Controller are created by this driver. If this parameter is not NULL and the first Device Path Node is not the End of Device Path Node, then only the handle for the child device specified by the first Device Path Node of RemainingDevicePath is created by this driver. If the first Device Path Node of RemainingDevicePath is the End of Device Path Node, no child handle is created by this driver.

Return values:

EFI_SUCCESS	The device was started.
EFI_DEVICE_ERROR	The device could not be started due to a device error.Currently not implemented.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
Others	The driver failded to start the device.

References DnsCreateService(), DnsDestroyService(), _DNS_SERVICE::ServiceBinding, and _DNS_SERVICE::Timer.

```
EFI_STATUS EFIAPI
Dns6DriverBindingStop (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     ControllerHandle,
  IN UINTN                          NumberOfChildren,
  IN EFI_HANDLE *                   ChildHandleBuffer
)
```

Stops a device controller or a bus controller.

The Stop() function is designed to be invoked from the EFI boot service DisconnectController(). As a result, much of the error checking on the parameters to Stop() has been moved into this common boot service. It is legal to call Stop() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE that was used on a previous call to this same driver's Start() function. 2. The first NumberOfChildren handles of ChildHandleBuffer must all be a valid EFI_HANDLE. In addition, all of these handles must have been created in this driver's Start() function, and the Start() function must have called OpenProtocol() on ControllerHandle with an Attribute of EFI_OPEN_PROTOCOL_BY_CHILD_CONTROLLER.

Parameters:

`[in]`	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
`[in]`	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
`[in]`	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
`[in]`	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer, _DNS_SERVICE::Dns6ChildrenList, DNS_SERVICE_FROM_THIS, DnsDestroyChildEntryInHandleBuffer(), DnsDestroyService(), gDnsControllerNameTable, DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren, and DNS_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding.

```
EFI_STATUS EFIAPI
Dns6DriverBindingSupported (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
)
```

Tests to see if this driver supports a given controller. If a child device is provided, it further tests to see if this driver supports creating a handle for the specified child device.

This function checks to see if the driver specified by This supports the device specified by ControllerHandle. Drivers will typically use the device path attached to ControllerHandle and/or the services from the bus I/O abstraction attached to ControllerHandle to determine if the driver supports ControllerHandle. This function may be called many times during platform initialization. In order to reduce boot times, the tests performed by this function must be very small, and take as little time as possible to execute. This function must not change the state of any hardware devices, and this function must be aware that the device specified by ControllerHandle may already be managed by the same driver or a different driver. This function must match its calls to AllocatePages() with FreePages(), AllocatePool() with FreePool(), and OpenProtocol() with CloseProtocol(). Because ControllerHandle may have been previously started by the same driver, if a protocol is already in the opened state, then it must not be closed with CloseProtocol(). This is required to guarantee the state of ControllerHandle is not modified by this function.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by the driver specified by This.
EFI_ACCESS_DENIED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by a different driver or an application that requires exclusive access. Currently not implemented.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

```
EFI_STATUS EFIAPI
Dns6ServiceBindingCreateChild (
  IN EFI_SERVICE_BINDING_PROTOCOL *   This,
  IN EFI_HANDLE *                     ChildHandle
)
```

Creates a child handle and installs a protocol.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

[in]	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
[in]	*ChildHandle*	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.
EFI_INVALID_PARAMETER	ChildHandle is NULL.
EFI_OUT_OF_RESOURCES	There are not enough resources availabe to create the child
other	The child handle was not created

References _DNS_INSTANCE::ChildHandle, _DNS_SERVICE::ConnectUdp, _DNS_INSTANCE::Dns6, _DNS_SERVICE::Dns6ChildrenList, _DNS_SERVICE::Dns6ChildrenNum, DNS_SERVICE_FROM_THIS, DnsCreateInstance(), DnsDestroyInstance(), gDns6DriverBinding, _DNS_INSTANCE::Link, and _DNS_INSTANCE::UdpIo.

```
EFI_STATUS EFIAPI
Dns6ServiceBindingDestroyChild ( IN EFI_SERVICE_BINDING_PROTOCOL * This,
                                 IN EFI_HANDLE ChildHandle
                               )
```

Destroys a child handle with a protocol installed on it.

The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

[in]	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
[in]	*ChildHandle*	Handle of the child to destroy

Return values:

EFI_SUCCES	The protocol was removed from ChildHandle.
EFI_UNSUPPORTED	ChildHandle does not support the protocol that is being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
EFI_ACCESS_DENIED	The protocol could not be removed from the ChildHandle because its services are being used.
other	The child handle was not destroyed

References _DNS_SERVICE::ConnectUdp, _DNS_SERVICE::Dns6ChildrenNum, DNS_INSTANCE_FROM_THIS_PROTOCOL6, DNS_SERVICE_FROM_THIS, DnsDestroyInstance(), gDns6DriverBinding, _DNS_INSTANCE::InDestroy, _DNS_INSTANCE::Link, _DNS_INSTANCE::Service, and _DNS_INSTANCE::UdpIo.

```
EFI_STATUS EFIAPI DnsConfigNullUdp ( IN UDP_IO * UdpIo,
                                     IN VOID * Context
                                   )
```

Config a NULL UDP that is used to keep the connection between UDP and DNS.

Just leave the Udp child unconfigured. When UDP is unloaded, DNS will be informed with DriverBinding Stop.

Parameters:

UdpIo The UDP_IO to configure
Context The opaque parameter to the callback

Return values:
EFI_SUCCESS It always return EFI_SUCCESS directly.

Referenced by DnsCreateInstance(), and DnsCreateService().

```
EFI_STATUS DnsCreateInstance ( IN DNS_SERVICE * Service,
                               OUT DNS_INSTANCE ** Instance
                             )
```

Create the DNS instance and initialize it.

Parameters:
[in] *Service* The pointer to the DNS service.
[out] *Instance* The pointer to the DNS instance.

Return values:
EFI_OUT_OF_RESOURCES Failed to allocate resources.
EFI_SUCCESS The DNS instance is created.

NicHandle

References _DNS_INSTANCE::Dns4, _DNS_INSTANCE::Dns4TxTokens, _DNS_INSTANCE::Dns6, _DNS_INSTANCE::Dns6TxTokens, DNS_INSTANCE_SIGNATURE, DNS_STATE_UNCONFIGED, DnsConfigNullUdp(), _DNS_INSTANCE::InDestroy, _DNS_INSTANCE::Link, mDns4Protocol, mDns6Protocol, _DNS_INSTANCE::Service, _DNS_INSTANCE::Signature, _DNS_INSTANCE::State, and _DNS_INSTANCE::UdpIo.

Referenced by Dns4ServiceBindingCreateChild(), and Dns6ServiceBindingCreateChild().

```
EFI_STATUS DnsCreateService ( IN EFI_HANDLE        Controller,
                              IN EFI_HANDLE        Image,
                              IN UINT8             IpVersion,
                              OUT DNS_SERVICE **   Service
                            )
```

Create then initialize a Dns service binding instance.

Parameters:
Controller The controller to install the DNS service binding on
Image The driver binding image of the DNS driver
IpVersion IpVersion for this service
Service The variable to receive the created service binding instance.

Return values:
EFI_OUT_OF_RESOURCES Failed to allocate resource to create the instance.
EFI_DEVICE_ERROR Failed to create a NULL UDP port to keep connection with UDP.

EFI_SUCCESS The service instance is created for the controller.

References _DNS_SERVICE::ConnectUdp, _DNS_SERVICE::ControllerHandle, _DNS_SERVICE::Dns4ChildrenList, _DNS_SERVICE::Dns4ChildrenNum, _DNS_SERVICE::Dns6ChildrenList, _DNS_SERVICE::Dns6ChildrenNum, DNS_SERVICE_SIGNATURE, DnsConfigNullUdp(), DnsOnTimerRetransmit(), _DNS_SERVICE::ImageHandle, _DNS_SERVICE::IpVersion, mDns4ServiceBinding, mDns6ServiceBinding, _DNS_SERVICE::ServiceBinding, _DNS_SERVICE::Signature, _DNS_SERVICE::Timer, and _DNS_SERVICE::TimerToGetMap.

Referenced by Dns4DriverBindingStart(), and Dns6DriverBindingStart().

EFI_STATUS EFIAPI DnsDestroyChildEntryInHandleBuffer (IN LIST_ENTRY * *Entry*, IN VOID * *Context*)

Callback function which provided by user to remove one node in NetDestroyLinkList process.

Parameters:

[in] *Entry* The entry to be removed.

[in] *Context* Pointer to the callback context corresponds to the Context in NetDestroyLinkList.

Return values:

EFI_SUCCESS The entry has been removed successfully.

Others Fail to remove the entry.

References _DNS_INSTANCE::ChildHandle, and DNS_INSTANCE_SIGNATURE.

Referenced by Dns4DriverBindingStop(), and Dns6DriverBindingStop().

VOID DnsDestroyInstance (IN DNS_INSTANCE * *Instance*)

Destroy the DNS instance and recycle the resources.

Parameters:

[in] *Instance* The pointer to the DNS instance.

References Dns4InstanceCancelToken(), and Dns6InstanceCancelToken().

Referenced by Dns4ServiceBindingCreateChild(), Dns4ServiceBindingDestroyChild(), Dns6ServiceBindingCreateChild(), and Dns6ServiceBindingDestroyChild().

VOID DnsDestroyService (IN DNS_SERVICE * *DnsSb*)

Release all the resource used the DNS service binding instance.

Parameters:

DnsSb The Dns service binding instance.

Referenced by Dns4DriverBindingStart(), Dns4DriverBindingStop(), Dns6DriverBindingStart(), and Dns6DriverBindingStop().

EFI_STATUS EFIAPI DnsDriverEntryPoint (IN EFI_HANDLE *ImageHandle*,
IN EFI_SYSTEM_TABLE * *SystemTable*
)

This is the declaration of an EFI image entry point. This entry point is the same for UEFI Applications, UEFI OS Loaders, and UEFI Drivers including both device drivers and bus drivers.

Parameters:

ImageHandle The firmware allocated handle for the UEFI image.
SystemTable A pointer to the EFI System Table.

Return values:

EFI_SUCCESS The operation completed successfully.
Others An unexpected error occurred.

References _DNS_DRIVER_DATA::Dns4CacheList, _DNS_DRIVER_DATA::Dns4ServerList, _DNS_DRIVER_DATA::Dns6CacheList, _DNS_DRIVER_DATA::Dns6ServerList, DnsOnTimerUpdate(), gDns4DriverBinding, gDns6DriverBinding, gDnsComponentName, gDnsComponentName2, and _DNS_DRIVER_DATA::Timer.

EFI_STATUS EFIAPI DnsUnload (IN EFI_HANDLE *ImageHandle*)

Unloads an image.

Parameters:

ImageHandle Handle that identifies the image to be unloaded.

Return values:

EFI_SUCCESS The image has been unloaded.
EFI_INVALID_PARAMETER ImageHandle is not a valid image handle.

References _DNS_DRIVER_DATA::Dns4CacheList, _DNS_DRIVER_DATA::Dns4ServerList, _DNS_DRIVER_DATA::Dns6CacheList, _DNS_DRIVER_DATA::Dns6ServerList, DNS6_CACHE::DnsCache, DNS4_CACHE::DnsCache, and _DNS_DRIVER_DATA::Timer.

Variable Documentation

DNS_DRIVER_DATA* mDriverData

Referenced by Dns4Configure(), Dns4GetModeData(), Dns4HostNameToIp(), Dns4UpdateDnsCache(), Dns6Configure(), Dns6GetModeData(), Dns6HostNameToIp(), Dns6UpdateDnsCache(), DnsOnTimerUpdate(), and ParseDnsResponse().

NetworkPkg/DnsDxe/DnsImpl.c File Reference

Functions

EFI_STATUS Dns4RemoveTokenEntry (IN NET_MAP *TokenMap, IN DNS4_TOKEN_ENTRY *TokenEntry)
EFI_STATUS Dns6RemoveTokenEntry (IN NET_MAP *TokenMap, IN DNS6_TOKEN_ENTRY *TokenEntry)
EFI_STATUS EFIAPI Dns4CancelTokens (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Arg)
EFI_STATUS EFIAPI Dns6CancelTokens (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Arg)
EFI_STATUS EFIAPI GetDns4TokenEntry (IN NET_MAP *TokensMap, IN EFI_DNS4_COMPLETION_TOKEN *Token, OUT DNS4_TOKEN_ENTRY **TokenEntry)
EFI_STATUS EFIAPI GetDns6TokenEntry (IN NET_MAP *TokensMap, IN EFI_DNS6_COMPLETION_TOKEN *Token, OUT DNS6_TOKEN_ENTRY **TokenEntry)
EFI_STATUS Dns4InstanceCancelToken (IN DNS_INSTANCE *Instance, IN EFI_DNS4_COMPLETION_TOKEN *Token)
EFI_STATUS Dns6InstanceCancelToken (IN DNS_INSTANCE *Instance, IN EFI_DNS6_COMPLETION_TOKEN *Token)
VOID Dns4CleanConfigure (IN OUT EFI_DNS4_CONFIG_DATA *Config)
VOID Dns6CleanConfigure (IN OUT EFI_DNS6_CONFIG_DATA *Config)
EFI_STATUS Dns4CopyConfigure (OUT EFI_DNS4_CONFIG_DATA *Dst, IN EFI_DNS4_CONFIG_DATA *Src)
EFI_STATUS Dns6CopyConfigure (OUT EFI_DNS6_CONFIG_DATA *Dst, IN EFI_DNS6_CONFIG_DATA *Src)
VOID EFIAPI DnsDummyExtFree (IN VOID *Arg)
BOOLEAN Dns4GetMapping (IN DNS_INSTANCE *Instance, IN UDP_IO *UdpIo, IN EFI_UDP4_CONFIG_DATA *UdpCfgData)
BOOLEAN Dns6GetMapping (IN DNS_INSTANCE *Instance, IN UDP_IO *UdpIo, IN EFI_UDP6_CONFIG_DATA *UdpCfgData)
EFI_STATUS Dns4ConfigUdp (IN DNS_INSTANCE *Instance, IN UDP_IO *UdpIo)
EFI_STATUS Dns6ConfigUdp (IN DNS_INSTANCE *Instance, IN UDP_IO *UdpIo)
EFI_STATUS EFIAPI UpdateDns4Cache (IN LIST_ENTRY *Dns4CacheList, IN BOOLEAN DeleteFlag, IN BOOLEAN Override, IN EFI_DNS4_CACHE_ENTRY DnsCacheEntry)
EFI_STATUS EFIAPI UpdateDns6Cache (IN LIST_ENTRY *Dns6CacheList, IN BOOLEAN DeleteFlag, IN BOOLEAN Override, IN EFI_DNS6_CACHE_ENTRY DnsCacheEntry)
EFI_STATUS EFIAPI AddDns4ServerIp (IN LIST_ENTRY *Dns4ServerList, IN EFI_IPv4_ADDRESS ServerIp)
EFI_STATUS EFIAPI AddDns6ServerIp (IN LIST_ENTRY *Dns6ServerList, IN EFI_IPv6_ADDRESS ServerIp)
BOOLEAN IsValidDnsResponse (IN NET_MAP *TokensMap, IN UINT16 Identification, IN UINT16 Type, OUT NET_MAP_ITEM **Item)
EFI_STATUS ParseDnsResponse (IN OUT DNS_INSTANCE *Instance, IN UINT8 *RxString, OUT BOOLEAN *Completed)
VOID EFIAPI DnsOnPacketReceived (NET_BUF *Packet, UDP_END_POINT *EndPoint, EFI_STATUS IoStatus, VOID *Context)
VOID EFIAPI DnsOnPacketSent (NET_BUF *Packet, UDP_END_POINT *EndPoint, EFI_STATUS IoStatus, VOID *Context)
EFI_STATUS DoDnsQuery (IN DNS_INSTANCE *Instance, IN NET_BUF *Packet)

EFI_STATUS ConstructDNSQueryIp (IN DNS_INSTANCE *Instance, IN CHAR16 *HostName, IN UINT16 Type, OUT NET_BUF **Packet)
EFI_STATUS DnsRetransmit (IN DNS_INSTANCE *Instance, IN NET_BUF *Packet)
VOID EFIAPI DnsOnTimerRetransmit (IN EFI_EVENT Event, IN VOID *Context)
VOID EFIAPI DnsOnTimerUpdate (IN EFI_EVENT Event, IN VOID *Context)

Detailed Description

DnsDxe support functions implementation.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI AddDns4ServerIp ( IN LIST_ENTRY *        Dns4ServerList,
                                    IN EFI_IPv4_ADDRESS    ServerIp
                                  )
```

Add Dns4 ServerIp to common list of addresses of all configured DNSv4 server.

Parameters:

Dns4ServerList	Common list of addresses of all configured DNSv4 server.
ServerIp	DNS server Ip.

Return values:

EFI_SUCCESS	Add Dns4 ServerIp to common list successfully.
Others	Failed to add Dns4 ServerIp to common list.

References DNS4_SERVER_IP::AllServerLink, and DNS4_SERVER_IP::Dns4ServerIp.

Referenced by Dns4Configure().

```
EFI_STATUS EFIAPI AddDns6ServerIp ( IN LIST_ENTRY *        Dns6ServerList,
                                    IN EFI_IPv6_ADDRESS    ServerIp
                                  )
```

Add Dns6 ServerIp to common list of addresses of all configured DNSv6 server.

Parameters:

Dns6ServerList	Common list of addresses of all configured DNSv6 server.
ServerIp	DNS server Ip.

Return values:

EFI_SUCCESS	Add Dns6 ServerIp to common list successfully.

Others Failed to add Dns6 ServerIp to common list.

References DNS6_SERVER_IP::AllServerLink, and DNS6_SERVER_IP::Dns6ServerIp.

Referenced by Dns6Configure().

```
EFI_STATUS ConstructDNSQueryIp ( IN DNS_INSTANCE *   Instance,
                                 IN CHAR16 *         HostName,
                                 IN UINT16           Type,
                                 OUT NET_BUF **      Packet
                               )
```

Construct the Packet to query Ip.

Parameters:

Instance The DNS instance
HostName Queried HostName
Type DNS query Type
Packet The packet for querying Ip

Return values:

EFI_SUCCESS The packet is constructed.
Others Failed to construct the Packet.

1 for header, 1 for tail.

References DNS_HEADER::AditionalNum, DNS_HEADER::AnswersNum, DNS_HEADER::AuthorityNum, _DNS_FLAGS::Bits, DNS_QUERY_SECTION::Class, DNS_CLASS_INET, DNS_DEFAULT_BLKSIZE, DNS_FLAGS_OPCODE_STANDARD, DNS_FLAGS_QR_QUERY, DnsDummyExtFree(), DNS_HEADER::Flags, DNS_HEADER::Identification, _DNS_FLAGS::OpCode, _DNS_FLAGS::QR, DNS_HEADER::QuestionsNum, _DNS_FLAGS::RD, DNS_QUERY_SECTION::Type, and _DNS_FLAGS::Uint16.

Referenced by Dns4HostNameToIp(), and Dns6HostNameToIp().

```
EFI_STATUS EFIAPI Dns4CancelTokens ( IN NET_MAP *        Map,
                                     IN
                                     NET_MAP_ITEM        Item,
                                     *
                                     IN VOID *           Arg
                                   )
```

This function cancle the token specified by Arg in the Map.

Parameters:

`[in]` *Map* Pointer to the NET_MAP.
`[in]` *Item* Pointer to the NET_MAP_ITEM.
`[in]` *Arg* Pointer to the token to be cancelled. If NULL, all the tokens in this Map will be cancelled. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS

The token is cancelled if Arg is NULL, or the token is not the same as that in the Item, if Arg is not NULL.

EFI_ABORTED Arg is not NULL, and the token specified by Arg is cancelled.

References Dns4RemoveTokenEntry(), and DNS4_TOKEN_ENTRY::Token.

Referenced by Dns4InstanceCancelToken().

VOID Dns4CleanConfigure (IN OUT EFI_DNS4_CONFIG_DATA * *Config*)

Free the resource related to the configure parameters.

Parameters:

Config The DNS configure data

Referenced by Dns4CopyConfigure().

EFI_STATUS Dns4ConfigUdp (IN DNS_INSTANCE * *Instance*,
IN UDP_IO * *UdpIo*
)

Configure the UDP.

Parameters:

Instance The DNS session

UdpIo The UDP_IO instance

Return values:

EFI_SUCCESS The UDP is successfully configured for the session.

References Dns4GetMapping(), and DNS_SERVER_PORT.

Referenced by Dns4Configure().

EFI_STATUS Dns4CopyConfigure (OUT EFI_DNS4_CONFIG_DATA * *Dst*,
IN EFI_DNS4_CONFIG_DATA * *Src*
)

Allocate memory for configure parameter such as timeout value for Dst, then copy the configure parameter from Src to Dst.

Parameters:

`[out]` *Dst* The destination DHCP configure data.

`[in]` *Src* The source DHCP configure data.

Return values:

EFI_OUT_OF_RESOURCES Failed to allocate memory.

EFI_SUCCESS The configure is copied.

References Dns4CleanConfigure().

Referenced by Dns4Configure(), and Dns4GetModeData().

```
BOOLEAN Dns4GetMapping ( IN DNS_INSTANCE *              Instance,
                         IN UDP_IO *                    UdpIo,
                         IN EFI_UDP4_CONFIG_DATA *      UdpCfgData
                       )
```

Poll the UDP to get the IP4 default address, which may be retrieved by DHCP.

The default time out value is 5 seconds. If IP has retrieved the default address, the UDP is reconfigured.

Parameters:

Instance	The DNS instance
UdpIo	The UDP_IO to poll
UdpCfgData	The UDP configure data to reconfigure the UDP_IO

Return values:

TRUE	The default address is retrieved and UDP is reconfigured.
FALSE	Some error occured.

References DNS_TIME_TO_GETMAP, and _DNS_SERVICE::TimerToGetMap.

Referenced by Dns4ConfigUdp().

```
EFI_STATUS Dns4InstanceCancelToken ( IN DNS_INSTANCE *                  Instance,
                                     IN EFI_DNS4_COMPLETION_TOKEN *     Token
                                   )
```

Cancel DNS4 tokens from the DNS4 instance.

Parameters:

[in] *Instance*	Pointer to the DNS instance context data.
[in] *Token*	Pointer to the token to be canceled. If NULL, all tokens in this instance will be cancelled. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The Token is cancelled.
EFI_NOT_FOUND	The Token is not found.

References Dns4CancelTokens(), and GetDns4TokenEntry().

Referenced by Dns4Cancel(), Dns4Configure(), and DnsDestroyInstance().

```
EFI_STATUS Dns4RemoveTokenEntry ( IN NET_MAP *              TokenMap,
                                  IN DNS4_TOKEN_ENTRY *     TokenEntry
                                )
```

Remove TokenEntry from TokenMap.

Parameters:

[in] *TokenMap*	All DNSv4 Token entrys.
[in] *TokenEntry*	TokenEntry need to be removed.

Return values:

EFI_SUCCESS	Remove TokenEntry from TokenMap sucessfully.

EFI_NOT_FOUND TokenEntry is not found in TokenMap.

Referenced by Dns4CancelTokens(), DnsOnTimerRetransmit(), and ParseDnsResponse().

```
EFI_STATUS EFIAPI Dns6CancelTokens ( IN NET_MAP *          Map,
                                     IN NET_MAP_ITEM *     Item,
                                     IN VOID *             Arg
                                   )
```

This function cancle the token specified by Arg in the Map.

Parameters:

[in] *Map* Pointer to the NET_MAP.

[in] *Item* Pointer to the NET_MAP_ITEM.

[in] *Arg* Pointer to the token to be cancelled. If NULL, all the tokens in this Map will be cancelled. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS The token is cancelled if Arg is NULL, or the token is not the same as that in the Item, if Arg is not NULL.

EFI_ABORTED Arg is not NULL, and the token specified by Arg is cancelled.

References Dns6RemoveTokenEntry(), and DNS6_TOKEN_ENTRY::Token.

Referenced by Dns6InstanceCancelToken().

```
VOID Dns6CleanConfigure ( IN OUT EFI_DNS6_CONFIG_DATA *   Config   )
```

Free the resource related to the configure parameters.

Parameters:

Config The DNS configure data

Referenced by Dns6CopyConfigure().

```
EFI_STATUS Dns6ConfigUdp ( IN DNS_INSTANCE *   Instance,
                           IN UDP_IO *         UdpIo
                         )
```

Configure the UDP.

Parameters:

Instance The DNS session

UdpIo The UDP_IO instance

Return values:

EFI_SUCCESS The UDP is successfully configured for the session.

References Dns6GetMapping(), and DNS_SERVER_PORT.

Referenced by Dns6Configure().

```
EFI_STATUS Dns6CopyConfigure ( OUT EFI_DNS6_CONFIG_DATA *  Dst,
                               IN EFI_DNS6_CONFIG_DATA *   Src
                             )
```

Allocate memory for configure parameter such as timeout value for Dst, then copy the configure parameter from Src to Dst.

Parameters:

[out]	*Dst*	The destination DHCP configure data.
[in]	*Src*	The source DHCP configure data.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_SUCCESS	The configure is copied.

References Dns6CleanConfigure().

Referenced by Dns6Configure(), and Dns6GetModeData().

```
BOOLEAN Dns6GetMapping ( IN DNS_INSTANCE *            Instance,
                         IN UDP_IO *                  UdpIo,
                         IN EFI_UDP6_CONFIG_DATA *    UdpCfgData
                       )
```

Configure the opened Udp6 instance until the corresponding Ip6 instance has been configured.

Parameters:

Instance	The DNS instance
UdpIo	The UDP_IO to poll
UdpCfgData	The UDP configure data to reconfigure the UDP_IO

Return values:

TRUE	Configure the Udp6 instance successfully.
FALSE	Some error occured.

References DNS_TIME_TO_GETMAP, and _DNS_SERVICE::TimerToGetMap.

Referenced by Dns6ConfigUdp().

```
EFI_STATUS Dns6InstanceCancelToken ( IN DNS_INSTANCE *                  Instance,
                                     IN EFI_DNS6_COMPLETION_TOKEN *     Token
                                   )
```

Cancel DNS6 tokens from the DNS6 instance.

Parameters:

[in]	*Instance*	Pointer to the DNS instance context data.
[in]	*Token*	Pointer to the token to be canceled. If NULL, all tokens in this instance will be cancelled. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The Token is cancelled.
EFI_NOT_FOUND	The Token is not found.

References Dns6CancelTokens(), and GetDns6TokenEntry().

Referenced by Dns6Cancel(), Dns6Configure(), and DnsDestroyInstance().

```
EFI_STATUS Dns6RemoveTokenEntry ( IN NET_MAP *              TokenMap,
                                  IN DNS6_TOKEN_ENTRY *     TokenEntry
                                )
```

Remove TokenEntry from TokenMap.

Parameters:

[in]	*TokenMap*	All DNSv6 Token entrys.
[in]	*TokenEntry*	TokenEntry need to be removed.

Return values:

EFI_SUCCESS	Remove TokenEntry from TokenMap sucessfully.
EFI_NOT_FOUND	TokenEntry is not found in TokenMap.

Referenced by Dns6CancelTokens(), DnsOnTimerRetransmit(), and ParseDnsResponse().

```
VOID EFIAPI DnsDummyExtFree ( IN VOID * Arg )
```

Callback of Dns packet. Does nothing.

Parameters:

Arg	The context.

Referenced by ConstructDNSQueryIp().

```
VOID EFIAPI DnsOnPacketReceived ( NET_BUF *          Packet,
                                  UDP_END_POINT *    EndPoint,
                                  EFI_STATUS         IoStatus,
                                  VOID *             Context
                                )
```

Parse response packet.

Parameters:

Packet	The packets received.
EndPoint	The local/remote UDP access point
IoStatus	The status of the UDP receive
Context	The opaque parameter to the function.

References DNS_INSTANCE_SIGNATURE, DnsOnPacketReceived(), ParseDnsResponse(), and _DNS_INSTANCE::UdpIo.

Referenced by DnsOnPacketReceived(), and DoDnsQuery().

```
VOID EFIAPI DnsOnPacketSent ( NET_BUF *          Packet,
                              UDP_END_POINT *    EndPoint,
                              EFI_STATUS         IoStatus,
                              VOID *             Context
                            )
```

Release the net buffer when packet is sent.

Parameters:

Packet	The packets received.
EndPoint	The local/remote UDP access point
IoStatus	The status of the UDP receive
Context	The opaque parameter to the function.

References _DNS_INSTANCE::Dns4TxTokens, _DNS_INSTANCE::Dns6TxTokens, DNS_INSTANCE_SIGNATURE, _DNS_SERVICE::IpVersion, DNS6_TOKEN_ENTRY::PacketToLive, DNS4_TOKEN_ENTRY::PacketToLive, _DNS_INSTANCE::Service, DNS6_TOKEN_ENTRY::Token, and DNS4_TOKEN_ENTRY::Token.

Referenced by DnsRetransmit(), and DoDnsQuery().

```
VOID EFIAPI DnsOnTimerRetransmit ( IN EFI_EVENT   Event,
                                   IN VOID *      Context
                                 )
```

The timer ticking function for the DNS services.

Parameters:

Event	The ticking event
Context	The DNS service instance

References _DNS_SERVICE::Dns4ChildrenList, Dns4RemoveTokenEntry(), _DNS_INSTANCE::Dns4TxTokens, _DNS_SERVICE::Dns6ChildrenList, Dns6RemoveTokenEntry(), _DNS_INSTANCE::Dns6TxTokens, DnsRetransmit(), _DNS_SERVICE::IpVersion, _DNS_INSTANCE::MaxRetry, DNS6_TOKEN_ENTRY::PacketToLive, DNS4_TOKEN_ENTRY::PacketToLive, DNS6_TOKEN_ENTRY::Token, and DNS4_TOKEN_ENTRY::Token.

Referenced by DnsCreateService().

```
VOID EFIAPI DnsOnTimerUpdate ( IN EFI_EVENT   Event,
                               IN VOID *      Context
                             )
```

The timer ticking function for the DNS driver.

Parameters:

Event	The ticking event
Context	NULL

References DNS6_CACHE::AllCacheLink, DNS4_CACHE::AllCacheLink, _DNS_DRIVER_DATA::Dns4CacheList, _DNS_DRIVER_DATA::Dns6CacheList, DNS6_CACHE::DnsCache, DNS4_CACHE::DnsCache, and mDriverData.

Referenced by DnsDriverEntryPoint().

```
EFI_STATUS DnsRetransmit ( IN DNS_INSTANCE *   Instance,
                           IN NET_BUF *        Packet
                         )
```

Retransmit the packet.

Parameters:

Instance The DNS instance
Packet Retransmit the packet

Return values:

EFI_SUCCESS The packet is retransmitted.
Others Failed to retransmit.

References DnsOnPacketSent().

Referenced by DnsOnTimerRetransmit().

```
EFI_STATUS DoDnsQuery ( IN DNS_INSTANCE *   Instance,
                        IN NET_BUF *        Packet
                      )
```

Query request information.

Parameters:

Instance The DNS instance
Packet The packet for querying request information.

Return values:

EFI_SUCCESS Query request information successfully.
Others Failed to query request information.

References DnsOnPacketReceived(), and DnsOnPacketSent().

Referenced by Dns4HostNameToIp(), and Dns6HostNameToIp().

```
EFI_STATUS EFIAPI GetDns4TokenEntry ( IN NET_MAP *                    TokensMap,
                                      IN EFI_DNS4_COMPLETION_TOKEN *  Token,
                                      OUT DNS4_TOKEN_ENTRY **         TokenEntry
                                    )
```

Get the TokenEntry from the TokensMap.

Parameters:

`[in]` *TokensMap* All DNSv4 Token entrys
`[in]` *Token* Pointer to the token to be get.
`[out]` *TokenEntry* Pointer to TokenEntry corresponding Token.

Return values:

EFI_SUCCESS Get the TokenEntry from the TokensMap sucessfully.

EFI_NOT_FOUND TokenEntry is not found in TokenMap.

Referenced by Dns4InstanceCancelToken().

```
EFI_STATUS EFIAPI GetDns6TokenEntry ( IN NET_MAP *                    TokensMap,
                                      IN EFI_DNS6_COMPLETION_TOKEN *  Token,
                                      OUT DNS6_TOKEN_ENTRY **         TokenEntry
                                    )
```

Get the TokenEntry from the TokensMap.

Parameters:

[in]	*TokensMap*	All DNSv6 Token entrys
[in]	*Token*	Pointer to the token to be get.
[out]	*TokenEntry*	Pointer to TokenEntry corresponding Token.

Return values:

EFI_SUCCESS	Get the TokenEntry from the TokensMap sucessfully.
EFI_NOT_FOUND	TokenEntry is not found in TokenMap.

Referenced by Dns6InstanceCancelToken().

```
BOOLEAN IsValidDnsResponse ( IN NET_MAP *          TokensMap,
                             IN UINT16             Identification,
                             IN UINT16             Type,
                             OUT NET_MAP_ITEM **   Item
                           )
```

Find out whether the response is valid or invalid.

Parameters:

TokensMap	All DNS transmittal Tokens entry.
Identification	Identification for queried packet.
Type	Type for queried packet.
Item	Return corresponding Token entry.

Return values:

TRUE	The response is valid.
FALSE	The response is invalid.

References DNS_HEADER::Identification, and DNS_QUERY_SECTION::Type.

Referenced by ParseDnsResponse().

```
EFI_STATUS ParseDnsResponse ( IN OUT DNS_INSTANCE *   Instance,
                              IN UINT8 *              RxString,
                              OUT BOOLEAN *           Completed
                            )
```

Parse Dns Response.

Parameters:

Instance The DNS instance
RxString Received buffer.
Completed Flag to indicate that Dns response is valid.

Return values:

EFI_SUCCESS Parse Dns Response successfully.
Others Failed to parse Dns Response.

References DNS_HEADER::AditionalNum, DNS_HEADER::AnswersNum, DNS_HEADER::AuthorityNum, _DNS_FLAGS::Bits, DNS_ANSWER_SECTION::Class, DNS_QUERY_SECTION::Class, DNS_ANSWER_SECTION::DataLength, _DNS_DRIVER_DATA::Dns4CacheList, Dns4RemoveTokenEntry(), _DNS_DRIVER_DATA::Dns6CacheList, Dns6RemoveTokenEntry(), DNS_CLASS_INET, DNS_FLAGS_QR_RESPONSE, DNS_FLAGS_RCODE_NO_ERROR, DNS_TYPE_A, DNS_TYPE_AAAA, DNS_HEADER::Flags, DNS_HEADER::Identification, IsValidDnsResponse(), mDriverData, _DNS_FLAGS::QR, DNS6_TOKEN_ENTRY::QueryHostName, DNS4_TOKEN_ENTRY::QueryHostName, DNS_HEADER::QuestionsNum, _DNS_FLAGS::RCode, DNS6_TOKEN_ENTRY::Token, DNS4_TOKEN_ENTRY::Token, DNS_ANSWER_SECTION::Ttl, DNS_ANSWER_SECTION::Type, DNS_QUERY_SECTION::Type, _DNS_FLAGS::Uint16, UpdateDns4Cache(), and UpdateDns6Cache().

Referenced by DnsOnPacketReceived().

```
EFI_STATUS EFIAPI UpdateDns4Cache ( IN LIST_ENTRY *                Dns4CacheList,
                                    IN BOOLEAN                     DeleteFlag,
                                    IN BOOLEAN                     Override,
                                    IN EFI_DNS4_CACHE_ENTRY        DnsCacheEntry
                                  )
```

Update Dns4 cache to shared list of caches of all DNSv4 instances.

Parameters:

Dns4CacheList All Dns4 cache list.
DeleteFlag If FALSE, this function is to add one entry to the DNS Cache. If TRUE, this function will delete matching DNS Cache entry.
Override If TRUE, the matching DNS cache entry will be overwritten with the supplied parameter. If FALSE, EFI_ACCESS_DENIED will be returned if the entry to be added is already exists.
DnsCacheEntry Entry Pointer to DNS Cache entry.

Return values:

EFI_SUCCESS Update Dns4 cache successfully.
Others Failed to update Dns4 cache.

References DNS4_CACHE::AllCacheLink, and DNS4_CACHE::DnsCache.

Referenced by Dns4UpdateDnsCache(), and ParseDnsResponse().

```
EFI_STATUS EFIAPI UpdateDns6Cache ( IN LIST_ENTRY *                Dns6CacheList,
                                    IN BOOLEAN                     DeleteFlag,
```

```
                                    IN BOOLEAN                  Override,
                                    IN EFI_DNS6_CACHE_ENTRY     DnsCacheEntry
                                  )
```

Update Dns6 cache to shared list of caches of all DNSv6 instances.

Parameters:

Dns6CacheList	All Dns6 cache list.
DeleteFlag	If FALSE, this function is to add one entry to the DNS Cache. If TRUE, this function will delete matching DNS Cache entry.
Override	If TRUE, the matching DNS cache entry will be overwritten with the supplied parameter. If FALSE, EFI_ACCESS_DENIED will be returned if the entry to be added is already exists.
DnsCacheEntry	Entry Pointer to DNS Cache entry.

Return values:

EFI_SUCCESS	Update Dns6 cache successfully.
Others	Failed to update Dns6 cache.

References DNS6_CACHE::AllCacheLink, and DNS6_CACHE::DnsCache.

Referenced by Dns6UpdateDnsCache(), and ParseDnsResponse().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/DnsDxe/DnsImpl.h File Reference

Data Structures

struct DNS4_CACHE
struct DNS6_CACHE
struct DNS4_SERVER_IP
struct DNS6_SERVER_IP
struct DNS4_TOKEN_ENTRY
struct DNS6_TOKEN_ENTRY
union _DNS_FLAGS
struct DNS_HEADER
struct DNS_QUERY_SECTION
struct DNS_ANSWER_SECTION

Defines

"memItemRight" valign="bottom">DNS_VERSION 0x00000000

#define DNS_SERVER_PORT 53
#define DNS_PROTOCOL_UDP EFI_IP_PROTO_UDP
#define DNS_PROTOCOL_TCP EFI_IP_PROTO_TCP
#define DNS_STATE_UNCONFIGED 0
#define DNS_STATE_CONFIGED 1
#define DNS_STATE_DESTROY 2
#define DNS_DEFAULT_TIMEOUT 2
#define DNS_DEFAULT_RETRY 3
#define DNS_DEFAULT_BLKSIZE 512
#define DNS_TIME_TO_GETMAP 5
#define DNS_FLAGS_QR_QUERY 0
#define DNS_FLAGS_QR_RESPONSE 1
#define DNS_FLAGS_OPCODE_STANDARD 0
#define DNS_FLAGS_OPCODE_INVERSE 1
#define DNS_FLAGS_OPCODE_SERVER_STATE 2
#define DNS_FLAGS_RCODE_NO_ERROR 0
#define DNS_FLAGS_RCODE_NAME_ERROR 3
#define DNS_TYPE_A 1
#define DNS_TYPE_NS 2
#define DNS_TYPE_CNAME 5
#define DNS_TYPE_PTR 12
#define DNS_TYPE_HINFO 13
#define DNS_TYPE_MX 15
#define DNS_TYPE_AAAA 28
#define DNS_TYPE_SRV_RR 33
#define DNS_TYPE_AXFR 252
#define DNS_TYPE_ANY 255
#define DNS_CLASS_INET 1
#define DNS4_DOMAIN L"in-addr.arpa"
#define DNS6_DOMAIN L"IP6.ARPA"

Typedefs

typedef union _DNS_FLAGS DNS_FLAGS

Functions

EFI_STATUS Dns4RemoveTokenEntry (IN NET_MAP *TokenMap, IN DNS4_TOKEN_ENTRY *TokenEntry)

EFI_STATUS Dns6RemoveTokenEntry (IN NET_MAP *TokenMap, IN DNS6_TOKEN_ENTRY *TokenEntry)

EFI_STATUS EFIAPI Dns4CancelTokens (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Arg)

EFI_STATUS EFIAPI Dns6CancelTokens (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Arg)

EFI_STATUS EFIAPI GetDns4TokenEntry (IN NET_MAP *TokensMap, IN EFI_DNS4_COMPLETION_TOKEN *Token, OUT DNS4_TOKEN_ENTRY **TokenEntry)

EFI_STATUS EFIAPI GetDns6TokenEntry (IN NET_MAP *TokensMap, IN EFI_DNS6_COMPLETION_TOKEN *Token, OUT DNS6_TOKEN_ENTRY **TokenEntry)

EFI_STATUS Dns4InstanceCancelToken (IN DNS_INSTANCE *Instance, IN EFI_DNS4_COMPLETION_TOKEN *Token)

EFI_STATUS Dns6InstanceCancelToken (IN DNS_INSTANCE *Instance, IN EFI_DNS6_COMPLETION_TOKEN *Token)

VOID Dns4CleanConfigure (IN OUT EFI_DNS4_CONFIG_DATA *Config)

VOID Dns6CleanConfigure (IN OUT EFI_DNS6_CONFIG_DATA *Config)

EFI_STATUS Dns4CopyConfigure (OUT EFI_DNS4_CONFIG_DATA *Dst, IN EFI_DNS4_CONFIG_DATA *Src)

EFI_STATUS Dns6CopyConfigure (OUT EFI_DNS6_CONFIG_DATA *Dst, IN EFI_DNS6_CONFIG_DATA *Src)

VOID EFIAPI DnsDummyExtFree (IN VOID *Arg)

BOOLEAN Dns4GetMapping (IN DNS_INSTANCE *Instance, IN UDP_IO *UdpIo, IN EFI_UDP4_CONFIG_DATA *UdpCfgData)

BOOLEAN Dns6GetMapping (IN DNS_INSTANCE *Instance, IN UDP_IO *UdpIo, IN EFI_UDP6_CONFIG_DATA *UdpCfgData)

EFI_STATUS Dns4ConfigUdp (IN DNS_INSTANCE *Instance, IN UDP_IO *UdpIo)

EFI_STATUS Dns6ConfigUdp (IN DNS_INSTANCE *Instance, IN UDP_IO *UdpIo)

EFI_STATUS EFIAPI UpdateDns4Cache (IN LIST_ENTRY *Dns4CacheList, IN BOOLEAN DeleteFlag, IN BOOLEAN Override, IN

EFI_DNS4_CACHE_ENTRY DnsCacheEntry)

EFI_STATUS EFIAPI UpdateDns6Cache (IN LIST_ENTRY *Dns6CacheList, IN BOOLEAN DeleteFlag, IN BOOLEAN Override, IN EFI_DNS6_CACHE_ENTRY DnsCacheEntry)

EFI_STATUS EFIAPI AddDns4ServerIp (IN LIST_ENTRY *Dns4ServerList, IN EFI_IPv4_ADDRESS ServerIp)

EFI_STATUS EFIAPI AddDns6ServerIp (IN LIST_ENTRY *Dns6ServerList, IN EFI_IPv6_ADDRESS ServerIp)

BOOLEAN IsValidDnsResponse (IN NET_MAP *TokensMap, IN UINT16 Identification, IN UINT16 Type, OUT NET_MAP_ITEM **Item)

EFI_STATUS ParseDnsResponse (IN OUT DNS_INSTANCE *Instance, IN UINT8 *RxString, OUT BOOLEAN *Completed)

VOID EFIAPI DnsOnPacketReceived (NET_BUF *Packet, UDP_END_POINT *EndPoint, EFI_STATUS IoStatus, VOID *Context)

VOID EFIAPI DnsOnPacketSent (NET_BUF *Packet, UDP_END_POINT *EndPoint, EFI_STATUS IoStatus, VOID *Context)

EFI_STATUS DoDnsQuery (IN DNS_INSTANCE *Instance, IN NET_BUF *Packet)

EFI_STATUS ConstructDNSQueryIp (IN DNS_INSTANCE *Instance, IN CHAR16 *HostName, IN UINT16 Type, OUT NET_BUF **Packet)

EFI_STATUS DnsRetransmit (IN DNS_INSTANCE *Instance, IN NET_BUF *Packet)

VOID EFIAPI DnsOnTimerRetransmit (IN EFI_EVENT Event, IN VOID *Context)

VOID EFIAPI DnsOnTimerUpdate (IN EFI_EVENT Event, IN VOID *Context)

EFI_STATUS EFIAPI Dns4GetModeData (IN EFI_DNS4_PROTOCOL *This, OUT EFI_DNS4_MODE_DATA *DnsModeData)

EFI_STATUS EFIAPI Dns4Configure (IN EFI_DNS4_PROTOCOL *This, IN EFI_DNS4_CONFIG_DATA *DnsConfigData)

EFI_STATUS EFIAPI Dns4HostNameToIp (IN EFI_DNS4_PROTOCOL *This, IN CHAR16 *HostName, IN EFI_DNS4_COMPLETION_TOKEN *Token)

EFI_STATUS EFIAPI Dns4IpToHostName (IN EFI_DNS4_PROTOCOL *This, IN EFI_IPv4_ADDRESS IpAddress, IN EFI_DNS4_COMPLETION_TOKEN *Token)

EFI_STATUS EFIAPI Dns4GeneralLookUp (IN EFI_DNS4_PROTOCOL *This, IN CHAR8 *QName, IN UINT16 QType, IN UINT16 QClass, IN EFI_DNS4_COMPLETION_TOKEN *Token)

EFI_STATUS EFIAPI Dns4UpdateDnsCache (IN

EFI_DNS4_PROTOCOL *This, IN BOOLEAN DeleteFlag, IN BOOLEAN Override, IN EFI_DNS4_CACHE_ENTRY DnsCacheEntry)
EFI_STATUS EFIAPI Dns4Poll (IN EFI_DNS4_PROTOCOL *This)
EFI_STATUS EFIAPI Dns4Cancel (IN EFI_DNS4_PROTOCOL *This, IN EFI_DNS4_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Dns6GetModeData (IN EFI_DNS6_PROTOCOL *This, OUT EFI_DNS6_MODE_DATA *DnsModeData)
EFI_STATUS EFIAPI Dns6Configure (IN EFI_DNS6_PROTOCOL *This, IN EFI_DNS6_CONFIG_DATA *DnsConfigData)
EFI_STATUS EFIAPI Dns6HostNameToIp (IN EFI_DNS6_PROTOCOL *This, IN CHAR16 *HostName, IN EFI_DNS6_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Dns6IpToHostName (IN EFI_DNS6_PROTOCOL *This, IN EFI_IPv6_ADDRESS IpAddress, IN EFI_DNS6_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Dns6GeneralLookUp (IN EFI_DNS6_PROTOCOL *This, IN CHAR8 *QName, IN UINT16 QType, IN UINT16 QClass, IN EFI_DNS6_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Dns6UpdateDnsCache (IN EFI_DNS6_PROTOCOL *This, IN BOOLEAN DeleteFlag, IN BOOLEAN Override, IN EFI_DNS6_CACHE_ENTRY DnsCacheEntry)
EFI_STATUS EFIAPI Dns6Poll (IN EFI_DNS6_PROTOCOL *This)
EFI_STATUS EFIAPI Dns6Cancel (IN EFI_DNS6_PROTOCOL *This, IN EFI_DNS6_COMPLETION_TOKEN *Token)

Variables

EFI_COMPONENT_NAME_PROTOCOL gDnsComponentName
EFI_COMPONENT_NAME2_PROTOCOL gDnsComponentName2
EFI_UNICODE_STRING_TABLE * gDnsControllerNameTable
EFI_DRIVER_BINDING_PROTOCOL gDns4DriverBinding
EFI_SERVICE_BINDING_PROTOCOL mDns4ServiceBinding
EFI_DNS4_PROTOCOL mDns4Protocol
EFI_DRIVER_BINDING_PROTOCOL gDns6DriverBinding
EFI_SERVICE_BINDING_PROTOCOL mDns6ServiceBinding
EFI_DNS6_PROTOCOL mDns6Protocol

Detailed Description

DnsDxe support functions implementation.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at

http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

#define DNS6_DOMAIN L"IP6.ARPA"

#define DNS_CLASS_INET 1

Referenced by ConstructDNSQueryIp(), and ParseDnsResponse().

#define DNS_DEFAULT_BLKSIZE 512

Referenced by ConstructDNSQueryIp().

#define DNS_DEFAULT_RETRY 3

Referenced by Dns4HostNameToIp(), and Dns6HostNameToIp().

#define DNS_DEFAULT_TIMEOUT 2

Referenced by Dns4HostNameToIp(), and Dns6HostNameToIp().

#define DNS_FLAGS_OPCODE_INVERSE 1

#define DNS_FLAGS_OPCODE_SERVER_STATE 2

#define DNS_FLAGS_OPCODE_STANDARD 0

Referenced by ConstructDNSQueryIp().

#define DNS_FLAGS_QR_QUERY 0

Referenced by ConstructDNSQueryIp().

#define DNS_FLAGS_QR_RESPONSE 1

Referenced by ParseDnsResponse().

#define DNS_FLAGS_RCODE_NAME_ERROR 3

#define DNS_FLAGS_RCODE_NO_ERROR 0

Referenced by ParseDnsResponse().

#define DNS_PROTOCOL_TCP EFI_IP_PROTO_TCP

#define DNS_PROTOCOL_UDP EFI_IP_PROTO_UDP

Referenced by Dns4Configure(), and Dns6Configure().

#define DNS_SERVER_PORT 53

Referenced by Dns4ConfigUdp(), and Dns6ConfigUdp().

#define DNS_STATE_CONFIGED 1

Referenced by Dns4Configure(), Dns4HostNameToIp(), Dns6Configure(), and Dns6HostNameToIp().

#define DNS_STATE_DESTROY 2

Referenced by Dns4Poll(), and Dns6Poll().

#define DNS_STATE_UNCONFIGED 0

Referenced by Dns4Cancel(), Dns4Configure(), Dns4GetModeData(), Dns4Poll(), Dns6Cancel(), Dns6Configure(), Dns6GetModeData(), Dns6Poll(), and DnsCreateInstance().

#define DNS_TIME_TO_GETMAP 5

Referenced by Dns4GetMapping(), Dns6GetMapping(), GetDns4ServerFromDhcp4(), and GetDns6ServerFromDhcp6().

#define DNS_TYPE_A 1

Referenced by Dns4HostNameToIp(), and ParseDnsResponse().

#define DNS_TYPE_AAAA 28

Referenced by Dns6HostNameToIp(), and ParseDnsResponse().

#define DNS_TYPE_ANY 255

#define DNS_TYPE_AXFR 252

#define DNS_TYPE_CNAME 5

#define DNS_TYPE_HINFO 13

#define DNS_TYPE_MX 15

#define DNS_TYPE_NS 2

#define DNS_TYPE_PTR 12

#define DNS_TYPE_SRV_RR 33

#define DNS_VERSION 0x00000000

Typedef Documentation

typedef union _DNS_FLAGS DNS_FLAGS

Function Documentation

EFI_STATUS EFIAPI AddDns4ServerIp (IN LIST_ENTRY * *Dns4ServerList*,
IN EFI_IPv4_ADDRESS *ServerIp*
)

Add Dns4 ServerIp to common list of addresses of all configured DNSv4 server.

Parameters:

Dns4ServerList	Common list of addresses of all configured DNSv4 server.
ServerIp	DNS server Ip.

Return values:

EFI_SUCCESS	Add Dns4 ServerIp to common list successfully.
Others	Failed to add Dns4 ServerIp to common list.

References DNS4_SERVER_IP::AllServerLink, and DNS4_SERVER_IP::Dns4ServerIp.

Referenced by Dns4Configure().

```
EFI_STATUS EFIAPI AddDns6ServerIp ( IN LIST_ENTRY *        Dns6ServerList,
                                    IN EFI_IPv6_ADDRESS    ServerIp
                                  )
```

Add Dns6 ServerIp to common list of addresses of all configured DNSv6 server.

Parameters:

Dns6ServerList	Common list of addresses of all configured DNSv6 server.
ServerIp	DNS server Ip.

Return values:

EFI_SUCCESS	Add Dns6 ServerIp to common list successfully.
Others	Failed to add Dns6 ServerIp to common list.

References DNS6_SERVER_IP::AllServerLink, and DNS6_SERVER_IP::Dns6ServerIp.

Referenced by Dns6Configure().

```
EFI_STATUS ConstructDNSQueryIp ( IN DNS_INSTANCE *   Instance,
                                 IN CHAR16 *         HostName,
                                 IN UINT16           Type,
                                 OUT NET_BUF **      Packet
                               )
```

Construct the Packet to query Ip.

Parameters:

Instance	The DNS instance
HostName	Queried HostName
Type	DNS query Type
Packet	The packet for querying Ip

Return values:

EFI_SUCCESS	The packet is constructed.
Others	Failed to construct the Packet.

1 for header, 1 for tail.

References DNS_HEADER::AditionalNum, DNS_HEADER::AnswersNum, DNS_HEADER::AuthorityNum, _DNS_FLAGS::Bits, DNS_QUERY_SECTION::Class, DNS_CLASS_INET, DNS_DEFAULT_BLKSIZE, DNS_FLAGS_OPCODE_STANDARD, DNS_FLAGS_QR_QUERY, DnsDummyExtFree(), DNS_HEADER::Flags, DNS_HEADER::Identification, _DNS_FLAGS::OpCode, _DNS_FLAGS::QR, DNS_HEADER::QuestionsNum, _DNS_FLAGS::RD, DNS_QUERY_SECTION::Type, and _DNS_FLAGS::Uint16.

Referenced by Dns4HostNameToIp(), and Dns6HostNameToIp().

```
EFI_STATUS EFIAPI Dns4Cancel ( IN EFI_DNS4_PROTOCOL *                This,
                               IN EFI_DNS4_COMPLETION_TOKEN *        Token
                             )
```

This function is used to abort a pending resolution request. After calling this function, Token.Status will be set

to EFI_ABORTED and then Token.

Parameters:

[in]	*This*	Pointer to EFI_DNS4_PROTOCOL instance.
[in]	*Token*	Pointer to a token that has been issued by EFI_DNS4_PROTOCOL.HostNameToIp(), EFI_DNS4_PROTOCOL.IpToHostName() or EFI_DNS4_PROTOCOL.GeneralLookup(). If NULL, all pending tokens are aborted.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This EFI DNS Protocol instance has not been started.
EFI_NOT_FOUND	When Token is not NULL, and the asynchronous DNS operation was not found in the transmit queue. It was either completed or was not issued by HostNameToIp(), IpToHostName() or GeneralLookup().

References Dns4InstanceCancelToken(), DNS_INSTANCE_FROM_THIS_PROTOCOL4, DNS_STATE_UNCONFIGED, and _DNS_INSTANCE::State.

```
EFI_STATUS EFIAPI Dns4CancelTokens ( IN NET_MAP *          Map,
                                     IN NET_MAP_ITEM *     Item,
                                     IN VOID *             Arg
                                   )
```

This function cancle the token specified by Arg in the Map.

Parameters:

[in]	*Map*	Pointer to the NET_MAP.
[in]	*Item*	Pointer to the NET_MAP_ITEM.
[in]	*Arg*	Pointer to the token to be cancelled. If NULL, all the tokens in this Map will be cancelled. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The token is cancelled if Arg is NULL, or the token is not the same as that in the Item, if Arg is not NULL.
EFI_ABORTED	Arg is not NULL, and the token specified by Arg is cancelled.

References Dns4RemoveTokenEntry(), and DNS4_TOKEN_ENTRY::Token.

Referenced by Dns4InstanceCancelToken().

```
VOID Dns4CleanConfigure ( IN OUT EFI_DNS4_CONFIG_DATA *   Config   )
```

Free the resource related to the configure parameters.

Parameters:

Config	The DNS configure data

Referenced by Dns4CopyConfigure().

```
EFI_STATUS Dns4ConfigUdp ( IN DNS_INSTANCE *   Instance,
                           IN UDP_IO *         UdpIo
                         )
```

Configure the UDP.

Parameters:

Instance The DNS session
UdpIo The UDP_IO instance

Return values:

EFI_SUCCESS The UDP is successfully configured for the session.

References Dns4GetMapping(), and DNS_SERVER_PORT.

Referenced by Dns4Configure().

```
EFI_STATUS EFIAPI Dns4Configure ( IN EFI_DNS4_PROTOCOL *        This,
                                  IN EFI_DNS4_CONFIG_DATA *     DnsConfigData
                                )
```

This function is used to configure DNS configuration data for this DNS instance.

Parameters:

`[in]` *This* Pointer to EFI_DNS4_PROTOCOL instance.
`[in]` *DnsConfigData* Pointer to caller-allocated buffer containing EFI_DNS4_CONFIG_DATA structure. If NULL, the driver will reinitialize the protocol instance to the unconfigured state.

Return values:

EFI_SUCCESS The operation completed successfully.
EFI_UNSUPPORTED The designated protocol is not supported.
EFI_OUT_OF_RESOURCES Failed to allocate needed resources.
EFI_INVALID_PARAMETER This is NULL. The StationIp address provided in DnsConfigData is not a valid unicast. DnsServerList is NULL while DnsServerListCount is not equal to Zero. DnsServerListCount is Zero while DnsServerListCount is not equal to NULL.
EFI_DEVICE_ERROR An unexpected system or network error occurred. The EFI DNSv4 Protocol instance is not configured.

References AddDns4ServerIp(), _DNS_INSTANCE::Dns4CfgData, Dns4ConfigUdp(), Dns4CopyConfigure(), Dns4InstanceCancelToken(), _DNS_DRIVER_DATA::Dns4ServerList, _DNS_INSTANCE::Dns4TxTokens, DNS_INSTANCE_FROM_THIS_PROTOCOL4, DNS_PROTOCOL_UDP, DNS_STATE_CONFIGED, DNS_STATE_UNCONFIGED, GetDns4ServerFromDhcp4(), _DNS_INSTANCE::MaxRetry, mDriverData, _DNS_INSTANCE::SessionDnsServer, _DNS_INSTANCE::State, and _DNS_INSTANCE::UdpIo.

```
EFI_STATUS Dns4CopyConfigure ( OUT EFI_DNS4_CONFIG_DATA *   Dst,
                               IN EFI_DNS4_CONFIG_DATA *    Src
                             )
```

Allocate memory for configure parameter such as timeout value for Dst, then copy the configure parameter from Src to Dst.

Parameters:

[out]	*Dst*	The destination DHCP configure data.
[in]	*Src*	The source DHCP configure data.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_SUCCESS	The configure is copied.

References Dns4CleanConfigure().

Referenced by Dns4Configure(), and Dns4GetModeData().

```
EFI_STATUS EFIAPI Dns4GeneralLookUp ( IN EFI_DNS4_PROTOCOL *              This,
                                      IN CHAR8 *                          QName,
                                      IN UINT16                           QType,
                                      IN UINT16                           QClass,
                                      IN EFI_DNS4_COMPLETION_TOKEN *      Token
                                    )
```

This function retrieves arbitrary information from the DNS. The caller supplies a QNAME, QTYPE, and QCLASS, and all of the matching RRs are returned. All RR content (e.g., Ttl) was returned. The caller need parse the returned RR to get required information. This function is optional.

Parameters:

[in]	*This*	Pointer to EFI_DNS4_PROTOCOL instance.
[in]	*QName*	Pointer to Query Name.
[in]	*QType*	Query Type.
[in]	*QClass*	Query Name.
[in]	*Token*	Point to the caller-allocated completion token to retrieve arbitrary information.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL. Token is NULL. Token.Event is NULL. QName is NULL.
EFI_NO_MAPPING	There's no source address is available for use.
EFI_ALREADY_STARTED	This Token is being used in another DNS session.
EFI_UNSUPPORTED	This function is not supported. Or the requested QType is not supported

```
BOOLEAN Dns4GetMapping ( IN DNS_INSTANCE *              Instance,
                         IN UDP_IO *                    UdpIo,
                         IN EFI_UDP4_CONFIG_DATA *      UdpCfgData
                       )
```

Poll the UDP to get the IP4 default address, which may be retrieved by DHCP.

The default time out value is 5 seconds. If IP has retrieved the default address, the UDP is reconfigured.

Parameters:

Instance	The DNS instance

UdpIo The UDP_IO to poll
UdpCfgData The UDP configure data to reconfigure the UDP_IO

Return values:
TRUE The default address is retrieved and UDP is reconfigured.
FALSE Some error occured.

References DNS_TIME_TO_GETMAP, and _DNS_SERVICE::TimerToGetMap.

Referenced by Dns4ConfigUdp().

```
EFI_STATUS EFIAPI Dns4GetModeData ( IN EFI_DNS4_PROTOCOL *        This,
                                    OUT EFI_DNS4_MODE_DATA *      DnsModeData
                                  )
```

This function is used to retrieve DNS mode data for this DNS instance.

Parameters:
`[in]` *This* Pointer to EFI_DNS4_PROTOCOL instance.
`[out]` *DnsModeData* Pointer to the caller-allocated storage for the EFI_DNS4_MODE_DATA structure.

Return values:
EFI_SUCCESS The operation completed successfully.
EFI_NOT_STARTED When DnsConfigData is queried, no configuration data is available because this instance has not been configured.
EFI_OUT_OF_RESOURCES Failed to allocate needed resources.
EFI_INVALID_PARAMETER This is NULL or DnsModeData is NULL.

References _DNS_DRIVER_DATA::Dns4CacheList, _DNS_INSTANCE::Dns4CfgData, Dns4CopyConfigure(), DNS4_SERVER_IP::Dns4ServerIp, _DNS_DRIVER_DATA::Dns4ServerList, DNS_INSTANCE_FROM_THIS_PROTOCOL4, DNS_STATE_UNCONFIGED, DNS4_CACHE::DnsCache, mDriverData, and _DNS_INSTANCE::State.

```
EFI_STATUS EFIAPI Dns4HostNameToIp ( IN EFI_DNS4_PROTOCOL *              This,
                                     IN CHAR16 *                          HostName,
                                     IN EFI_DNS4_COMPLETION_TOKEN *       Token
                                   )
```

The function is used to translate the host name to host IP address. A type A query is used to get the one or more IP addresses for this host.

Parameters:
`[in]` *This* Pointer to EFI_DNS4_PROTOCOL instance.
`[in]` *HostName* Pointer to caller-supplied buffer containing Host name to be translated. This buffer contains 16 bit characters but these are translated to ASCII for use with DNSv4 server and there is no requirement for driver to support non-ASCII Unicode characters.
`[in]` *Token* Pointer to the caller-allocated completion token to return at the completion of the process to translate host name to host address.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL. Token is NULL. Token.Event is.NULL HostName is NULL
EFI_NO_MAPPING	There's no source address is available for use.
EFI_NOT_STARTED	This instance has not been started.

References ConstructDNSQueryIp(), _DNS_DRIVER_DATA::Dns4CacheList, _DNS_INSTANCE::Dns4CfgData, _DNS_INSTANCE::Dns4TxTokens, DNS_DEFAULT_RETRY, DNS_DEFAULT_TIMEOUT, DNS_INSTANCE_FROM_THIS_PROTOCOL4, DNS_STATE_CONFIGED, DNS_TYPE_A, DNS4_CACHE::DnsCache, DoDnsQuery(), _DNS_INSTANCE::MaxRetry, mDriverData, DNS4_TOKEN_ENTRY::PacketToLive, DNS4_TOKEN_ENTRY::QueryHostName, _DNS_INSTANCE::State, and DNS4_TOKEN_ENTRY::Token.

```
EFI_STATUS Dns4InstanceCancelToken ( IN DNS_INSTANCE *                    Instance,
                                     IN EFI_DNS4_COMPLETION_TOKEN *       Token
                                   )
```

Cancel DNS4 tokens from the DNS4 instance.

Parameters:

[in]	*Instance*	Pointer to the DNS instance context data.
[in]	*Token*	Pointer to the token to be canceled. If NULL, all tokens in this instance will be cancelled. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The Token is cancelled.
EFI_NOT_FOUND	The Token is not found.

References Dns4CancelTokens(), and GetDns4TokenEntry().

Referenced by Dns4Cancel(), Dns4Configure(), and DnsDestroyInstance().

```
EFI_STATUS EFIAPI Dns4IpToHostName ( IN EFI_DNS4_PROTOCOL *               This,
                                     IN EFI_IPv4_ADDRESS                  IpAddress,
                                     IN EFI_DNS4_COMPLETION_TOKEN *       Token
                                   )
```

The function is used to translate the host address to host name. A type PTR query is used to get the primary name of the host.

Parameters:

[in]	*This*	Pointer to EFI_DNS4_PROTOCOL instance.
[in]	*IpAddress*	IP address.
[in]	*Token*	Pointer to the caller-allocated completion used token to translate host address to host name.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL. Token is NULL. Token.Event is NULL. IpAddress is not valid IP address.

EFI_NO_MAPPING	There's no source address is available for use.
EFI_NOT_STARTED	This instance has not been started.
EFI_UNSUPPORTED	This function is not supported.

EFI_STATUS EFIAPI Dns4Poll (IN EFI_DNS4_PROTOCOL * *This*)

This function can be used by network drivers and applications to increase the rate that data packets are moved between the communications device and the transmit and receive queues. In some systems, the periodic timer event in the managed network driver may not poll the underlying communications device fast enough to transmit and/or receive all data packets without missing incoming packets or dropping outgoing packets.

Parameters:

`[in]` *This* Pointer to EFI_DNS4_PROTOCOL instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This EFI DNS Protocol instance has not been started.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_TIMEOUT	Data was dropped out of the transmit and/or receive queue. Consider increasing the polling rate.

References DNS_INSTANCE_FROM_THIS_PROTOCOL4, DNS_STATE_DESTROY, DNS_STATE_UNCONFIGED, _DNS_INSTANCE::State, and _DNS_INSTANCE::UdpIo.

EFI_STATUS Dns4RemoveTokenEntry (IN NET_MAP * *TokenMap*,
IN DNS4_TOKEN_ENTRY * *TokenEntry*
)

Remove TokenEntry from TokenMap.

Parameters:

`[in]` *TokenMap* All DNSv4 Token entrys.

`[in]` *TokenEntry* TokenEntry need to be removed.

Return values:

EFI_SUCCESS	Remove TokenEntry from TokenMap sucessfully.
EFI_NOT_FOUND	TokenEntry is not found in TokenMap.

Referenced by Dns4CancelTokens(), DnsOnTimerRetransmit(), and ParseDnsResponse().

EFI_STATUS EFIAPI Dns4UpdateDnsCache (IN EFI_DNS4_PROTOCOL * *This*,
IN BOOLEAN *DeleteFlag*,
IN BOOLEAN *Override*,
IN EFI_DNS4_CACHE_ENTRY *DnsCacheEntry*
)

This function is used to add/delete/modify DNS cache entry. DNS cache can be normally dynamically updated after the DNS resolve succeeds. This function provided capability to manually add/delete/modify the DNS cache.

Parameters:

[in]	*This*	Pointer to EFI_DNS4_PROTOCOL instance.
[in]	*DeleteFlag*	If FALSE, this function is to add one entry to the DNS Cache. If TRUE, this function will delete matching DNS Cache entry.
[in]	*Override*	If TRUE, the matching DNS cache entry will be overwritten with the supplied parameter. If FALSE, EFI_ACCESS_DENIED will be returned if the entry to be added is already exists.
[in]	*DnsCacheEntry*	Pointer to DNS Cache entry.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	This is NULL. DnsCacheEntry.HostName is NULL. DnsCacheEntry.IpAddress is NULL. DnsCacheEntry.Timeout is zero.
EFI_ACCESS_DENIED	The DNS cache entry already exists and Override is not TRUE.

References _DNS_DRIVER_DATA::Dns4CacheList, mDriverData, and UpdateDns4Cache().

```
EFI_STATUS EFIAPI Dns6Cancel ( IN EFI_DNS6_PROTOCOL *            This,
                               IN EFI_DNS6_COMPLETION_TOKEN *    Token
                             )
```

This function is used to abort a pending resolution request. After calling this function, Token.Status will be set to EFI_ABORTED and then Token.

Parameters:

[in]	*This*	Pointer to EFI_DNS6_PROTOCOL instance.
[in]	*Token*	Pointer to a token that has been issued by EFI_DNS6_PROTOCOL.HostNameToIp(), EFI_DNS6_PROTOCOL.IpToHostName() or EFI_DNS6_PROTOCOL.GeneralLookup(). If NULL, all pending tokens are aborted.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This EFI DNS Protocol instance has not been started.
EFI_NOT_FOUND	When Token is not NULL, and the asynchronous DNS operation was not found in the transmit queue. It was either completed or was not issued by HostNameToIp(), IpToHostName() or GeneralLookup().

References Dns6InstanceCancelToken(), DNS_INSTANCE_FROM_THIS_PROTOCOL6, DNS_STATE_UNCONFIGED, and _DNS_INSTANCE::State.

```
EFI_STATUS EFIAPI Dns6CancelTokens ( IN NET_MAP *          Map,
                                     IN
                                     NET_MAP_ITEM          Item,
                                     *
                                     IN VOID *             Arg
                                   )
```

This function cancle the token specified by Arg in the Map.

Parameters:

[in]	*Map*	Pointer to the NET_MAP.
[in]	*Item*	Pointer to the NET_MAP_ITEM.

[in] *Arg* Pointer to the token to be cancelled. If NULL, all the tokens in this Map will be cancelled. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS The token is cancelled if Arg is NULL, or the token is not the same as that in the Item, if Arg is not NULL.

EFI_ABORTED Arg is not NULL, and the token specified by Arg is cancelled.

References Dns6RemoveTokenEntry(), and DNS6_TOKEN_ENTRY::Token.

Referenced by Dns6InstanceCancelToken().

VOID Dns6CleanConfigure (IN OUT EFI_DNS6_CONFIG_DATA * *Config*)

Free the resource related to the configure parameters.

Parameters:

Config The DNS configure data

Referenced by Dns6CopyConfigure().

EFI_STATUS Dns6ConfigUdp (IN DNS_INSTANCE * *Instance*,
IN UDP_IO * *UdpIo*
)

Configure the UDP.

Parameters:

Instance The DNS session

UdpIo The UDP_IO instance

Return values:

EFI_SUCCESS The UDP is successfully configured for the session.

References Dns6GetMapping(), and DNS_SERVER_PORT.

Referenced by Dns6Configure().

EFI_STATUS EFIAPI Dns6Configure (IN EFI_DNS6_PROTOCOL * *This*,
IN EFI_DNS6_CONFIG_DATA * *DnsConfigData*
)

The function is used to set and change the configuration data for this EFI DNSv6 Protocol driver instance. Reset the DNS instance if DnsConfigData is NULL.

Parameters:

[in] *This* Pointer to EFI_DNS6_PROTOCOL instance.

[in] *DnsConfigData* Pointer to the configuration data structure. All associated storage to be allocated and released by caller.

Return values:

EFI_SUCCESS The operation completed successfully.

EFI_UNSUPPORTED The designated protocol is not supported.

EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL. The StationIp address provided in DnsConfigData is not a valid unicast. DnsServerList is NULL while DnsServerListCount is not equal to Zero. DnsServerListCount is Zero while DnsServerList is not equal to NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI DNSv6 Protocol instance is not configured.

References AddDns6ServerIp(), _DNS_SERVICE::ControllerHandle, _DNS_INSTANCE::Dns6CfgData, Dns6ConfigUdp(), Dns6CopyConfigure(), Dns6InstanceCancelToken(), _DNS_DRIVER_DATA::Dns6ServerList, _DNS_INSTANCE::Dns6TxTokens, DNS_INSTANCE_FROM_THIS_PROTOCOL6, DNS_PROTOCOL_UDP, DNS_STATE_CONFIGED, DNS_STATE_UNCONFIGED, GetDns6ServerFromDhcp6(), _DNS_SERVICE::ImageHandle, _DNS_INSTANCE::MaxRetry, mDriverData, _DNS_INSTANCE::Service, _DNS_INSTANCE::SessionDnsServer, _DNS_INSTANCE::State, and _DNS_INSTANCE::UdpIo.

```
EFI_STATUS Dns6CopyConfigure ( OUT EFI_DNS6_CONFIG_DATA *  Dst,
                               IN EFI_DNS6_CONFIG_DATA *   Src
                             )
```

Allocate memory for configure parameter such as timeout value for Dst, then copy the configure parameter from Src to Dst.

Parameters:

[out]	*Dst*	The destination DHCP configure data.
[in]	*Src*	The source DHCP configure data.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_SUCCESS	The configure is copied.

References Dns6CleanConfigure().

Referenced by Dns6Configure(), and Dns6GetModeData().

```
EFI_STATUS EFIAPI Dns6GeneralLookUp ( IN EFI_DNS6_PROTOCOL *              This,
                                      IN CHAR8 *                          QName,
                                      IN UINT16                           QType,
                                      IN UINT16                           QClass,
                                      IN EFI_DNS6_COMPLETION_TOKEN *      Token
                                    )
```

This function retrieves arbitrary information from the DNS. The caller supplies a QNAME, QTYPE, and QCLASS, and all of the matching RRs are returned. All RR content (e.g., Ttl) was returned. The caller need parse the returned RR to get required information. This function is optional.

Parameters:

[in]	*This*	Pointer to EFI_DNS6_PROTOCOL instance.
[in]	*QName*	Pointer to Query Name.
[in]	*QType*	Query Type.
[in]	*QClass*	Query Name.
[in]	*Token*	Point to the caller-allocated completion token to retrieve arbitrary information.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL. Token is NULL. Token.Event is NULL. QName is NULL.
EFI_NO_MAPPING	There's no source address is available for use.
EFI_NOT_STARTED	This instance has not been started.
EFI_UNSUPPORTED	This function is not supported. Or the requested QType is not supported

```
BOOLEAN Dns6GetMapping ( IN DNS_INSTANCE *            Instance,
                         IN UDP_IO *                  UdpIo,
                         IN EFI_UDP6_CONFIG_DATA *    UdpCfgData
                       )
```

Configure the opened Udp6 instance until the corresponding Ip6 instance has been configured.

Parameters:

Instance	The DNS instance
UdpIo	The UDP_IO to poll
UdpCfgData	The UDP configure data to reconfigure the UDP_IO

Return values:

TRUE	Configure the Udp6 instance successfully.
FALSE	Some error occured.

References DNS_TIME_TO_GETMAP, and _DNS_SERVICE::TimerToGetMap.

Referenced by Dns6ConfigUdp().

```
EFI_STATUS EFIAPI Dns6GetModeData ( IN EFI_DNS6_PROTOCOL *      This,
                                    OUT EFI_DNS6_MODE_DATA *    DnsModeData
                                  )
```

This function is used to retrieve DNS mode data for this DNS instance.

Parameters:

`[in]`	*This*	Pointer to EFI_DNS6_PROTOCOL instance.
`[out]`	*DnsModeData*	Pointer to the caller-allocated storage for the EFI_DNS6_MODE_DATA structure.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_STARTED	When DnsConfigData is queried, no configuration data is available because this instance has not been configured.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL or DnsModeData is NULL.

References _DNS_DRIVER_DATA::Dns6CacheList, _DNS_INSTANCE::Dns6CfgData, Dns6CopyConfigure(), DNS6_SERVER_IP::Dns6ServerIp, _DNS_DRIVER_DATA::Dns6ServerList, DNS_INSTANCE_FROM_THIS_PROTOCOL6, DNS_STATE_UNCONFIGED, DNS6_CACHE::DnsCache, mDriverData, and _DNS_INSTANCE::State.

```
EFI_STATUS EFIAPI Dns6HostNameToIp ( IN EFI_DNS6_PROTOCOL *              This,
                                     IN CHAR16 *                         HostName,
                                     IN EFI_DNS6_COMPLETION_TOKEN *      Token
                                   )
```

The function is used to translate the host name to host IP address. A type AAAA query is used to get the one or more IPv6 addresses for this host.

Parameters:

[in]	*This*	Pointer to EFI_DNS6_PROTOCOL instance.
[in]	*HostName*	Pointer to caller-supplied buffer containing Host name to be translated. This buffer contains 16 bit characters but these are translated to ASCII for use with DNSv4 server and there is no requirement for driver to support non-ASCII Unicode characters.
[in]	*Token*	Pointer to the caller-allocated completion token to return at the completion of the process to translate host name to host address.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL. Token is NULL. Token.Event is.NULL HostName is NULL
EFI_NO_MAPPING	There's no source address is available for use.
EFI_NOT_STARTED	This instance has not been started.

References ConstructDNSQueryIp(), _DNS_DRIVER_DATA::Dns6CacheList, _DNS_INSTANCE::Dns6CfgData, _DNS_INSTANCE::Dns6TxTokens, DNS_DEFAULT_RETRY, DNS_DEFAULT_TIMEOUT, DNS_INSTANCE_FROM_THIS_PROTOCOL6, DNS_STATE_CONFIGED, DNS_TYPE_AAAA, DNS6_CACHE::DnsCache, DoDnsQuery(), _DNS_INSTANCE::MaxRetry, mDriverData, DNS6_TOKEN_ENTRY::PacketToLive, DNS6_TOKEN_ENTRY::QueryHostName, _DNS_INSTANCE::State, and DNS6_TOKEN_ENTRY::Token.

```
EFI_STATUS Dns6InstanceCancelToken ( IN DNS_INSTANCE *                   Instance,
                                     IN EFI_DNS6_COMPLETION_TOKEN *      Token
                                   )
```

Cancel DNS6 tokens from the DNS6 instance.

Parameters:

[in]	*Instance*	Pointer to the DNS instance context data.
[in]	*Token*	Pointer to the token to be canceled. If NULL, all tokens in this instance will be cancelled. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The Token is cancelled.
EFI_NOT_FOUND	The Token is not found.

References Dns6CancelTokens(), and GetDns6TokenEntry().

Referenced by Dns6Cancel(), Dns6Configure(), and DnsDestroyInstance().

```
EFI_STATUS EFIAPI Dns6IpToHostName ( IN EFI_DNS6_PROTOCOL *              This,
```

```
                                    IN EFI_IPv6_ADDRESS                 IpAddress,
                                    IN EFI_DNS6_COMPLETION_TOKEN *      Token
                                  )
```

The function is used to translate the host address to host name. A type PTR query is used to get the primary name of the host.

Parameters:

[in]	*This*	Pointer to EFI_DNS6_PROTOCOL instance.
[in]	*IpAddress*	IP address.
[in]	*Token*	Pointer to the caller-allocated completion used token to translate host address to host name.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL. Token is NULL. Token.Event is NULL. IpAddress is not valid IP address.
EFI_NO_MAPPING	There's no source address is available for use.
EFI_NOT_STARTED	This instance has not been started.
EFI_UNSUPPORTED	This function is not supported.

```
EFI_STATUS EFIAPI Dns6Poll ( IN EFI_DNS6_PROTOCOL *  This )
```

This function can be used by network drivers and applications to increase the rate that data packets are moved between the communications device and the transmit and receive queues. In some systems, the periodic timer event in the managed network driver may not poll the underlying communications device fast enough to transmit and/or receive all data packets without missing incoming packets or dropping outgoing packets.

Parameters:

[in]	*This*	Pointer to EFI_DNS6_PROTOCOL instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This EFI DNS Protocol instance has not been started.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_TIMEOUT	Data was dropped out of the transmit and/or receive queue. Consider increasing the polling rate.

References DNS_INSTANCE_FROM_THIS_PROTOCOL6, DNS_STATE_DESTROY, DNS_STATE_UNCONFIGED, _DNS_INSTANCE::State, and _DNS_INSTANCE::UdpIo.

```
EFI_STATUS Dns6RemoveTokenEntry ( IN NET_MAP *                TokenMap,
                                  IN DNS6_TOKEN_ENTRY *       TokenEntry
                                )
```

Remove TokenEntry from TokenMap.

Parameters:

[in]	*TokenMap*	All DNSv6 Token entrys.
[in]	*TokenEntry*	TokenEntry need to be removed.

Return values:

EFI_SUCCESS	Remove TokenEntry from TokenMap sucessfully.
EFI_NOT_FOUND	TokenEntry is not found in TokenMap.

Referenced by Dns6CancelTokens(), DnsOnTimerRetransmit(), and ParseDnsResponse().

```
EFI_STATUS EFIAPI Dns6UpdateDnsCache ( IN EFI_DNS6_PROTOCOL *      This,
                                       IN BOOLEAN                  DeleteFlag,
                                       IN BOOLEAN                  Override,
                                       IN EFI_DNS6_CACHE_ENTRY     DnsCacheEntry
                                     )
```

This function is used to add/delete/modify DNS cache entry. DNS cache can be normally dynamically updated after the DNS resolve succeeds. This function provided capability to manually add/delete/modify the DNS cache.

Parameters:

[in]	*This*	Pointer to EFI_DNS6_PROTOCOL instance.
[in]	*DeleteFlag*	If FALSE, this function is to add one entry to the DNS Cache. If TRUE, this function will delete matching DNS Cache entry.
[in]	*Override*	If TRUE, the matching DNS cache entry will be overwritten with the supplied parameter. If FALSE, EFI_ACCESS_DENIED will be returned if the entry to be added is already exists.
[in]	*DnsCacheEntry*	Pointer to DNS Cache entry.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	This is NULL. DnsCacheEntry.HostName is NULL. DnsCacheEntry.IpAddress is NULL. DnsCacheEntry.Timeout is zero.
EFI_ACCESS_DENIED	The DNS cache entry already exists and Override is not TRUE.

References _DNS_DRIVER_DATA::Dns6CacheList, mDriverData, and UpdateDns6Cache().

```
VOID EFIAPI DnsDummyExtFree ( IN VOID * Arg )
```

Callback of Dns packet. Does nothing.

Parameters:

Arg The context.

Referenced by ConstructDNSQueryIp().

```
VOID EFIAPI DnsOnPacketReceived ( NET_BUF *          Packet,
                                  UDP_END_POINT *    EndPoint,
                                  EFI_STATUS         IoStatus,
                                  VOID *             Context
                                )
```

Parse response packet.

Parameters:

Packet The packets received.
EndPoint The local/remote UDP access point
IoStatus The status of the UDP receive
Context The opaque parameter to the function.

References DNS_INSTANCE_SIGNATURE, DnsOnPacketReceived(), ParseDnsResponse(), and _DNS_INSTANCE::UdpIo.

Referenced by DnsOnPacketReceived(), and DoDnsQuery().

```
VOID EFIAPI DnsOnPacketSent ( NET_BUF *          Packet,
                              UDP_END_POINT *    EndPoint,
                              EFI_STATUS         IoStatus,
                              VOID *             Context
                            )
```

Release the net buffer when packet is sent.

Parameters:
Packet The packets received.
EndPoint The local/remote UDP access point
IoStatus The status of the UDP receive
Context The opaque parameter to the function.

References _DNS_INSTANCE::Dns4TxTokens, _DNS_INSTANCE::Dns6TxTokens, DNS_INSTANCE_SIGNATURE, _DNS_SERVICE::IpVersion, DNS6_TOKEN_ENTRY::PacketToLive, DNS4_TOKEN_ENTRY::PacketToLive, _DNS_INSTANCE::Service, DNS6_TOKEN_ENTRY::Token, and DNS4_TOKEN_ENTRY::Token.

Referenced by DnsRetransmit(), and DoDnsQuery().

```
VOID EFIAPI DnsOnTimerRetransmit ( IN EFI_EVENT   Event,
                                   IN VOID *      Context
                                 )
```

The timer ticking function for the DNS service.

Parameters:
Event The ticking event
Context The DNS service instance

The timer ticking function for the DNS services.

Parameters:
Event The ticking event
Context The DNS service instance

References _DNS_SERVICE::Dns4ChildrenList, Dns4RemoveTokenEntry(), _DNS_INSTANCE::Dns4TxTokens, _DNS_SERVICE::Dns6ChildrenList, Dns6RemoveTokenEntry(), _DNS_INSTANCE::Dns6TxTokens, DnsRetransmit(), _DNS_SERVICE::IpVersion, _DNS_INSTANCE::MaxRetry, DNS6_TOKEN_ENTRY::PacketToLive, DNS4_TOKEN_ENTRY::PacketToLive, DNS6_TOKEN_ENTRY::Token, and

DNS4_TOKEN_ENTRY::Token.

Referenced by DnsCreateService().

```
VOID EFIAPI DnsOnTimerUpdate ( IN EFI_EVENT  Event,
                               IN VOID *     Context
                             )
```

The timer ticking function for the DNS driver.

Parameters:

Event The ticking event
Context NULL

References DNS6_CACHE::AllCacheLink, DNS4_CACHE::AllCacheLink, _DNS_DRIVER_DATA::Dns4CacheList, _DNS_DRIVER_DATA::Dns6CacheList, DNS6_CACHE::DnsCache, DNS4_CACHE::DnsCache, and mDriverData.

Referenced by DnsDriverEntryPoint().

```
EFI_STATUS DnsRetransmit ( IN DNS_INSTANCE *  Instance,
                           IN NET_BUF *       Packet
                         )
```

Retransmit the packet.

Parameters:

Instance The DNS instance
Packet Retransmit the packet

Return values:

EFI_SUCCESS The packet is retransmitted.
Others Failed to retransmit.

References DnsOnPacketSent().

Referenced by DnsOnTimerRetransmit().

```
EFI_STATUS DoDnsQuery ( IN DNS_INSTANCE *  Instance,
                        IN NET_BUF *       Packet
                      )
```

Query request information.

Parameters:

Instance The DNS instance
Packet The packet for querying request information.

Return values:

EFI_SUCCESS Query request information successfully.
Others Failed to query request information.

References DnsOnPacketReceived(), and DnsOnPacketSent().

Referenced by Dns4HostNameToIp(), and Dns6HostNameToIp().

```
EFI_STATUS EFIAPI GetDns4TokenEntry ( IN NET_MAP *                    TokensMap,
                                      IN EFI_DNS4_COMPLETION_TOKEN *  Token,
                                      OUT DNS4_TOKEN_ENTRY **         TokenEntry
                                    )
```

Get the TokenEntry from the TokensMap.

Parameters:

[in]	*TokensMap*	All DNSv4 Token entrys
[in]	*Token*	Pointer to the token to be get.
[out]	*TokenEntry*	Pointer to TokenEntry corresponding Token.

Return values:

EFI_SUCCESS	Get the TokenEntry from the TokensMap sucessfully.
EFI_NOT_FOUND	TokenEntry is not found in TokenMap.

Referenced by Dns4InstanceCancelToken().

```
EFI_STATUS EFIAPI GetDns6TokenEntry ( IN NET_MAP *                    TokensMap,
                                      IN EFI_DNS6_COMPLETION_TOKEN *  Token,
                                      OUT DNS6_TOKEN_ENTRY **         TokenEntry
                                    )
```

Get the TokenEntry from the TokensMap.

Parameters:

[in]	*TokensMap*	All DNSv6 Token entrys
[in]	*Token*	Pointer to the token to be get.
[out]	*TokenEntry*	Pointer to TokenEntry corresponding Token.

Return values:

EFI_SUCCESS	Get the TokenEntry from the TokensMap sucessfully.
EFI_NOT_FOUND	TokenEntry is not found in TokenMap.

Referenced by Dns6InstanceCancelToken().

```
BOOLEAN IsValidDnsResponse ( IN NET_MAP *          TokensMap,
                             IN UINT16             Identification,
                             IN UINT16             Type,
                             OUT NET_MAP_ITEM **   Item
                           )
```

Find out whether the response is valid or invalid.

Parameters:

TokensMap	All DNS transmittal Tokens entry.
Identification	Identification for queried packet.

Type Type for queried packet.
Item Return corresponding Token entry.

Return values:
TRUE The response is valid.
FALSE The response is invalid.

References DNS_HEADER::Identification, and DNS_QUERY_SECTION::Type.

Referenced by ParseDnsResponse().

```
EFI_STATUS ParseDnsResponse ( IN OUT DNS_INSTANCE *   Instance,
                              IN UINT8 *              RxString,
                              OUT BOOLEAN *           Completed
                            )
```

Parse Dns Response.

Parameters:
Instance The DNS instance
RxString Received buffer.
Completed Flag to indicate that Dns response is valid.

Return values:
EFI_SUCCESS Parse Dns Response successfully.
Others Failed to parse Dns Response.

References DNS_HEADER::AditionalNum, DNS_HEADER::AnswersNum, DNS_HEADER::AuthorityNum, _DNS_FLAGS::Bits, DNS_ANSWER_SECTION::Class, DNS_QUERY_SECTION::Class, DNS_ANSWER_SECTION::DataLength, _DNS_DRIVER_DATA::Dns4CacheList, Dns4RemoveTokenEntry(), _DNS_DRIVER_DATA::Dns6CacheList, Dns6RemoveTokenEntry(), DNS_CLASS_INET, DNS_FLAGS_QR_RESPONSE, DNS_FLAGS_RCODE_NO_ERROR, DNS_TYPE_A, DNS_TYPE_AAAA, DNS_HEADER::Flags, DNS_HEADER::Identification, IsValidDnsResponse(), mDriverData, _DNS_FLAGS::QR, DNS6_TOKEN_ENTRY::QueryHostName, DNS4_TOKEN_ENTRY::QueryHostName, DNS_HEADER::QuestionsNum, _DNS_FLAGS::RCode, DNS6_TOKEN_ENTRY::Token, DNS4_TOKEN_ENTRY::Token, DNS_ANSWER_SECTION::Ttl, DNS_ANSWER_SECTION::Type, DNS_QUERY_SECTION::Type, _DNS_FLAGS::Uint16, UpdateDns4Cache(), and UpdateDns6Cache().

Referenced by DnsOnPacketReceived().

```
EFI_STATUS EFIAPI UpdateDns4Cache ( IN LIST_ENTRY *              Dns4CacheList,
                                    IN BOOLEAN                   DeleteFlag,
                                    IN BOOLEAN                   Override,
                                    IN EFI_DNS4_CACHE_ENTRY      DnsCacheEntry
                                  )
```

Update Dns4 cache to shared list of caches of all DNSv4 instances.

Parameters:
Dns4CacheList All Dns4 cache list.

DeleteFlag If FALSE, this function is to add one entry to the DNS Cache. If TRUE, this function will delete matching DNS Cache entry.
Override If TRUE, the matching DNS cache entry will be overwritten with the supplied parameter. If FALSE, EFI_ACCESS_DENIED will be returned if the entry to be added is already exists.
DnsCacheEntry Entry Pointer to DNS Cache entry.

Return values:
EFI_SUCCESS Update Dns4 cache successfully.
Others Failed to update Dns4 cache.

References DNS4_CACHE::AllCacheLink, and DNS4_CACHE::DnsCache.

Referenced by Dns4UpdateDnsCache(), and ParseDnsResponse().

```
EFI_STATUS EFIAPI UpdateDns6Cache ( IN LIST_ENTRY *                Dns6CacheList,
                                    IN BOOLEAN                     DeleteFlag,
                                    IN BOOLEAN                     Override,
                                    IN EFI_DNS6_CACHE_ENTRY        DnsCacheEntry
                                  )
```

Update Dns6 cache to shared list of caches of all DNSv6 instances.

Parameters:
Dns6CacheList All Dns6 cache list.
DeleteFlag If FALSE, this function is to add one entry to the DNS Cache. If TRUE, this function will delete matching DNS Cache entry.
Override If TRUE, the matching DNS cache entry will be overwritten with the supplied parameter. If FALSE, EFI_ACCESS_DENIED will be returned if the entry to be added is already exists.
DnsCacheEntry Entry Pointer to DNS Cache entry.

Return values:
EFI_SUCCESS Update Dns6 cache successfully.
Others Failed to update Dns6 cache.

References DNS6_CACHE::AllCacheLink, and DNS6_CACHE::DnsCache.

Referenced by Dns6UpdateDnsCache(), and ParseDnsResponse().

Variable Documentation

EFI_DRIVER_BINDING_PROTOCOL gDns4DriverBinding

Referenced by Dns4ServiceBindingCreateChild(), Dns4ServiceBindingDestroyChild(), and DnsDriverEntryPoint().

EFI_DRIVER_BINDING_PROTOCOL gDns6DriverBinding

Referenced by Dns6ServiceBindingCreateChild(), Dns6ServiceBindingDestroyChild(), and DnsDriverEntryPoint().

EFI_COMPONENT_NAME_PROTOCOL gDnsComponentName
Component Name Protocol instance

Referenced by DnsComponentNameGetControllerName(), DnsComponentNameGetDriverName(), DnsDriverEntryPoint(), UpdateDns4Name(), and UpdateDns6Name().

EFI_COMPONENT_NAME2_PROTOCOL gDnsComponentName2
Component Name 2 Protocol instance

Referenced by DnsDriverEntryPoint(), UpdateDns4Name(), and UpdateDns6Name().

EFI_UNICODE_STRING_TABLE* gDnsControllerNameTable

Referenced by Dns4DriverBindingStop(), and Dns6DriverBindingStop().

EFI_DNS4_PROTOCOL mDns4Protocol

Referenced by DnsCreateInstance().

EFI_SERVICE_BINDING_PROTOCOL mDns4ServiceBinding

Referenced by DnsCreateService().

EFI_DNS6_PROTOCOL mDns6Protocol

Referenced by DnsCreateInstance().

EFI_SERVICE_BINDING_PROTOCOL mDns6ServiceBinding

Referenced by DnsCreateService().

NetworkPkg/DnsDxe/DnsProtocol.c File Reference

Functions

EFI_STATUS EFIAPI Dns4GetModeData (IN EFI_DNS4_PROTOCOL *This, OUT EFI_DNS4_MODE_DATA *DnsModeData)
EFI_STATUS EFIAPI Dns4Configure (IN EFI_DNS4_PROTOCOL *This, IN EFI_DNS4_CONFIG_DATA *DnsConfigData)
EFI_STATUS EFIAPI Dns4HostNameToIp (IN EFI_DNS4_PROTOCOL *This, IN CHAR16 *HostName, IN EFI_DNS4_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Dns4IpToHostName (IN EFI_DNS4_PROTOCOL *This, IN EFI_IPv4_ADDRESS IpAddress, IN EFI_DNS4_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Dns4GeneralLookUp (IN EFI_DNS4_PROTOCOL *This, IN CHAR8 *QName, IN UINT16 QType, IN UINT16 QClass, IN EFI_DNS4_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Dns4UpdateDnsCache (IN EFI_DNS4_PROTOCOL *This, IN BOOLEAN DeleteFlag, IN BOOLEAN Override, IN EFI_DNS4_CACHE_ENTRY DnsCacheEntry)
EFI_STATUS EFIAPI Dns4Poll (IN EFI_DNS4_PROTOCOL *This)
EFI_STATUS EFIAPI Dns4Cancel (IN EFI_DNS4_PROTOCOL *This, IN EFI_DNS4_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Dns6GetModeData (IN EFI_DNS6_PROTOCOL *This, OUT EFI_DNS6_MODE_DATA *DnsModeData)
EFI_STATUS EFIAPI Dns6Configure (IN EFI_DNS6_PROTOCOL *This, IN EFI_DNS6_CONFIG_DATA *DnsConfigData)
EFI_STATUS EFIAPI Dns6HostNameToIp (IN EFI_DNS6_PROTOCOL *This, IN CHAR16 *HostName, IN EFI_DNS6_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Dns6IpToHostName (IN EFI_DNS6_PROTOCOL *This, IN EFI_IPv6_ADDRESS IpAddress, IN EFI_DNS6_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Dns6GeneralLookUp (IN EFI_DNS6_PROTOCOL *This, IN CHAR8 *QName, IN UINT16 QType, IN UINT16 QClass, IN EFI_DNS6_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Dns6UpdateDnsCache (IN EFI_DNS6_PROTOCOL *This, IN BOOLEAN DeleteFlag, IN BOOLEAN Override, IN EFI_DNS6_CACHE_ENTRY DnsCacheEntry)
EFI_STATUS EFIAPI Dns6Poll (IN EFI_DNS6_PROTOCOL *This)
EFI_STATUS EFIAPI Dns6Cancel (IN EFI_DNS6_PROTOCOL *This, IN EFI_DNS6_COMPLETION_TOKEN *Token)

Variables

EFI_DNS4_PROTOCOL mDns4Protocol
EFI_DNS6_PROTOCOL mDns6Protocol

Detailed Description

Implementation of EFI_DNS4_PROTOCOL and EFI_DNS6_PROTOCOL interfaces.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI Dns4Cancel ( IN EFI_DNS4_PROTOCOL *              This,
                               IN EFI_DNS4_COMPLETION_TOKEN *      Token
                             )
```

This function is used to abort a pending resolution request. After calling this function, Token.Status will be set to EFI_ABORTED and then Token.

Parameters:

[in]	*This*	Pointer to EFI_DNS4_PROTOCOL instance.
[in]	*Token*	Pointer to a token that has been issued by EFI_DNS4_PROTOCOL.HostNameToIp(), EFI_DNS4_PROTOCOL.IpToHostName() or EFI_DNS4_PROTOCOL.GeneralLookup(). If NULL, all pending tokens are aborted.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This EFI DNS Protocol instance has not been started.
EFI_NOT_FOUND	When Token is not NULL, and the asynchronous DNS operation was not found in the transmit queue. It was either completed or was not issued by HostNameToIp(), IpToHostName() or GeneralLookup().

References Dns4InstanceCancelToken(), DNS_INSTANCE_FROM_THIS_PROTOCOL4, DNS_STATE_UNCONFIGED, and _DNS_INSTANCE::State.

```
EFI_STATUS EFIAPI Dns4Configure ( IN EFI_DNS4_PROTOCOL *        This,
                                  IN EFI_DNS4_CONFIG_DATA *     DnsConfigData
                                )
```

This function is used to configure DNS configuration data for this DNS instance.

Parameters:

[in]	*This*	Pointer to EFI_DNS4_PROTOCOL instance.
[in]	*DnsConfigData*	Pointer to caller-allocated buffer containing EFI_DNS4_CONFIG_DATA structure. If NULL, the driver will reinitialize the protocol instance to the unconfigured state.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_UNSUPPORTED	The designated protocol is not supported.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL. The StationIp address provided in DnsConfigData is not a valid unicast. DnsServerList is NULL while

	DnsServerListCount is not equal to Zero. DnsServerListCount is Zero while DnsServerListCount is not equal to NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI DNSv4 Protocol instance is not configured.

References AddDns4ServerIp(), _DNS_INSTANCE::Dns4CfgData, Dns4ConfigUdp(), Dns4CopyConfigure(), Dns4InstanceCancelToken(), _DNS_DRIVER_DATA::Dns4ServerList, _DNS_INSTANCE::Dns4TxTokens, DNS_INSTANCE_FROM_THIS_PROTOCOL4, DNS_PROTOCOL_UDP, DNS_STATE_CONFIGED, DNS_STATE_UNCONFIGED, GetDns4ServerFromDhcp4(), _DNS_INSTANCE::MaxRetry, mDriverData, _DNS_INSTANCE::SessionDnsServer, _DNS_INSTANCE::State, and _DNS_INSTANCE::UdpIo.

```
EFI_STATUS EFIAPI Dns4GeneralLookUp ( IN EFI_DNS4_PROTOCOL *              This,
                                      IN CHAR8 *                          QName,
                                      IN UINT16                           QType,
                                      IN UINT16                           QClass,
                                      IN EFI_DNS4_COMPLETION_TOKEN *      Token
                                    )
```

This function retrieves arbitrary information from the DNS. The caller supplies a QNAME, QTYPE, and QCLASS, and all of the matching RRs are returned. All RR content (e.g., Ttl) was returned. The caller need parse the returned RR to get required information. This function is optional.

Parameters:

[in]	*This*	Pointer to EFI_DNS4_PROTOCOL instance.
[in]	*QName*	Pointer to Query Name.
[in]	*QType*	Query Type.
[in]	*QClass*	Query Name.
[in]	*Token*	Point to the caller-allocated completion token to retrieve arbitrary information.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL. Token is NULL. Token.Event is NULL. QName is NULL.
EFI_NO_MAPPING	There's no source address is available for use.
EFI_ALREADY_STARTED	This Token is being used in another DNS session.
EFI_UNSUPPORTED	This function is not supported. Or the requested QType is not supported

```
EFI_STATUS EFIAPI Dns4GetModeData ( IN EFI_DNS4_PROTOCOL *        This,
                                    OUT EFI_DNS4_MODE_DATA *      DnsModeData
                                  )
```

This function is used to retrieve DNS mode data for this DNS instance.

Parameters:

[in]	*This*	Pointer to EFI_DNS4_PROTOCOL instance.
[out]	*DnsModeData*	Pointer to the caller-allocated storage for the EFI_DNS4_MODE_DATA structure.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_STARTED	When DnsConfigData is queried, no configuration data is available because this instance has not been configured.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL or DnsModeData is NULL.

References _DNS_DRIVER_DATA::Dns4CacheList, _DNS_INSTANCE::Dns4CfgData, Dns4CopyConfigure(), DNS4_SERVER_IP::Dns4ServerIp, _DNS_DRIVER_DATA::Dns4ServerList, DNS_INSTANCE_FROM_THIS_PROTOCOL4, DNS_STATE_UNCONFIGED, DNS4_CACHE::DnsCache, mDriverData, and _DNS_INSTANCE::State.

```
EFI_STATUS EFIAPI Dns4HostNameToIp ( IN EFI_DNS4_PROTOCOL *              This,
                                     IN CHAR16 *                         HostName,
                                     IN EFI_DNS4_COMPLETION_TOKEN *      Token
                                   )
```

The function is used to translate the host name to host IP address. A type A query is used to get the one or more IP addresses for this host.

Parameters:

`[in]`	*This*	Pointer to EFI_DNS4_PROTOCOL instance.
`[in]`	*HostName*	Pointer to caller-supplied buffer containing Host name to be translated. This buffer contains 16 bit characters but these are translated to ASCII for use with DNSv4 server and there is no requirement for driver to support non-ASCII Unicode characters.
`[in]`	*Token*	Pointer to the caller-allocated completion token to return at the completion of the process to translate host name to host address.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL. Token is NULL. Token.Event is.NULL HostName is NULL
EFI_NO_MAPPING	There's no source address is available for use.
EFI_NOT_STARTED	This instance has not been started.

References ConstructDNSQueryIp(), _DNS_DRIVER_DATA::Dns4CacheList, _DNS_INSTANCE::Dns4CfgData, _DNS_INSTANCE::Dns4TxTokens, DNS_DEFAULT_RETRY, DNS_DEFAULT_TIMEOUT, DNS_INSTANCE_FROM_THIS_PROTOCOL4, DNS_STATE_CONFIGED, DNS_TYPE_A, DNS4_CACHE::DnsCache, DoDnsQuery(), _DNS_INSTANCE::MaxRetry, mDriverData, DNS4_TOKEN_ENTRY::PacketToLive, DNS4_TOKEN_ENTRY::QueryHostName, _DNS_INSTANCE::State, and DNS4_TOKEN_ENTRY::Token.

```
EFI_STATUS EFIAPI Dns4IpToHostName ( IN EFI_DNS4_PROTOCOL *              This,
                                     IN EFI_IPv4_ADDRESS                 IpAddress,
                                     IN EFI_DNS4_COMPLETION_TOKEN *      Token
                                   )
```

The function is used to translate the host address to host name. A type PTR query is used to get the primary name of the host.

Parameters:

[in]	*This*	Pointer to EFI_DNS4_PROTOCOL instance.
[in]	*IpAddress*	IP address.
[in]	*Token*	Pointer to the caller-allocated completion used token to translate host address to host name.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL. Token is NULL. Token.Event is NULL. IpAddress is not valid IP address.
EFI_NO_MAPPING	There's no source address is available for use.
EFI_NOT_STARTED	This instance has not been started.
EFI_UNSUPPORTED	This function is not supported.

EFI_STATUS EFIAPI Dns4Poll (IN EFI_DNS4_PROTOCOL * *This*)

This function can be used by network drivers and applications to increase the rate that data packets are moved between the communications device and the transmit and receive queues. In some systems, the periodic timer event in the managed network driver may not poll the underlying communications device fast enough to transmit and/or receive all data packets without missing incoming packets or dropping outgoing packets.

Parameters:

[in]	*This*	Pointer to EFI_DNS4_PROTOCOL instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This EFI DNS Protocol instance has not been started.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_TIMEOUT	Data was dropped out of the transmit and/or receive queue. Consider increasing the polling rate.

References DNS_INSTANCE_FROM_THIS_PROTOCOL4, DNS_STATE_DESTROY, DNS_STATE_UNCONFIGED, _DNS_INSTANCE::State, and _DNS_INSTANCE::UdpIo.

```
EFI_STATUS EFIAPI Dns4UpdateDnsCache ( IN EFI_DNS4_PROTOCOL *      This,
                                       IN BOOLEAN                  DeleteFlag,
                                       IN BOOLEAN                  Override,
                                       IN EFI_DNS4_CACHE_ENTRY     DnsCacheEntry
                                     )
```

This function is used to add/delete/modify DNS cache entry. DNS cache can be normally dynamically updated after the DNS resolve succeeds. This function provided capability to manually add/delete/modify the DNS cache.

Parameters:

[in]	*This*	Pointer to EFI_DNS4_PROTOCOL instance.
[in]	*DeleteFlag*	If FALSE, this function is to add one entry to the DNS Cache. If TRUE, this function will delete matching DNS Cache entry.
[in]	*Override*	If TRUE, the matching DNS cache entry will be overwritten with the supplied parameter. If FALSE, EFI_ACCESS_DENIED will be returned if

the entry to be added is already exists.

[in] *DnsCacheEntry* Pointer to DNS Cache entry.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	This is NULL. DnsCacheEntry.HostName is NULL. DnsCacheEntry.IpAddress is NULL. DnsCacheEntry.Timeout is zero.
EFI_ACCESS_DENIED	The DNS cache entry already exists and Override is not TRUE.

References _DNS_DRIVER_DATA::Dns4CacheList, mDriverData, and UpdateDns4Cache().

```
EFI_STATUS EFIAPI Dns6Cancel ( IN EFI_DNS6_PROTOCOL *              This,
                               IN EFI_DNS6_COMPLETION_TOKEN *      Token
                             )
```

This function is used to abort a pending resolution request. After calling this function, Token.Status will be set to EFI_ABORTED and then Token.

Parameters:

[in] *This* Pointer to EFI_DNS6_PROTOCOL instance.

[in] *Token* Pointer to a token that has been issued by EFI_DNS6_PROTOCOL.HostNameToIp(), EFI_DNS6_PROTOCOL.IpToHostName() or EFI_DNS6_PROTOCOL.GeneralLookup(). If NULL, all pending tokens are aborted.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This EFI DNS Protocol instance has not been started.
EFI_NOT_FOUND	When Token is not NULL, and the asynchronous DNS operation was not found in the transmit queue. It was either completed or was not issued by HostNameToIp(), IpToHostName() or GeneralLookup().

References Dns6InstanceCancelToken(), DNS_INSTANCE_FROM_THIS_PROTOCOL6, DNS_STATE_UNCONFIGED, and _DNS_INSTANCE::State.

```
EFI_STATUS EFIAPI Dns6Configure ( IN EFI_DNS6_PROTOCOL *        This,
                                  IN EFI_DNS6_CONFIG_DATA *     DnsConfigData
                                )
```

The function is used to set and change the configuration data for this EFI DNSv6 Protocol driver instance. Reset the DNS instance if DnsConfigData is NULL.

Parameters:

[in] *This* Pointer to EFI_DNS6_PROTOCOL instance.

[in] *DnsConfigData* Pointer to the configuration data structure. All associated storage to be allocated and released by caller.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_UNSUPPORTED	The designated protocol is not supported.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	

	This is NULL. The StationIp address provided in DnsConfigData is not a valid unicast. DnsServerList is NULL while DnsServerListCount is not equal to Zero. DnsServerListCount is Zero while DnsServerList is not equal to NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI DNSv6 Protocol instance is not configured.

References AddDns6ServerIp(), _DNS_SERVICE::ControllerHandle, _DNS_INSTANCE::Dns6CfgData, Dns6ConfigUdp(), Dns6CopyConfigure(), Dns6InstanceCancelToken(), _DNS_DRIVER_DATA::Dns6ServerList, _DNS_INSTANCE::Dns6TxTokens, DNS_INSTANCE_FROM_THIS_PROTOCOL6, DNS_PROTOCOL_UDP, DNS_STATE_CONFIGED, DNS_STATE_UNCONFIGED, GetDns6ServerFromDhcp6(), _DNS_SERVICE::ImageHandle, _DNS_INSTANCE::MaxRetry, mDriverData, _DNS_INSTANCE::Service, _DNS_INSTANCE::SessionDnsServer, _DNS_INSTANCE::State, and _DNS_INSTANCE::UdpIo.

```
EFI_STATUS EFIAPI Dns6GeneralLookUp ( IN EFI_DNS6_PROTOCOL *              This,
                                      IN CHAR8 *                          QName,
                                      IN UINT16                           QType,
                                      IN UINT16                           QClass,
                                      IN EFI_DNS6_COMPLETION_TOKEN *      Token
                                    )
```

This function retrieves arbitrary information from the DNS. The caller supplies a QNAME, QTYPE, and QCLASS, and all of the matching RRs are returned. All RR content (e.g., Ttl) was returned. The caller need parse the returned RR to get required information. This function is optional.

Parameters:

[in]	*This*	Pointer to EFI_DNS6_PROTOCOL instance.
[in]	*QName*	Pointer to Query Name.
[in]	*QType*	Query Type.
[in]	*QClass*	Query Name.
[in]	*Token*	Point to the caller-allocated completion token to retrieve arbitrary information.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL. Token is NULL. Token.Event is NULL. QName is NULL.
EFI_NO_MAPPING	There's no source address is available for use.
EFI_NOT_STARTED	This instance has not been started.
EFI_UNSUPPORTED	This function is not supported. Or the requested QType is not supported

```
EFI_STATUS EFIAPI Dns6GetModeData ( IN EFI_DNS6_PROTOCOL *        This,
                                    OUT EFI_DNS6_MODE_DATA *      DnsModeData
                                  )
```

This function is used to retrieve DNS mode data for this DNS instance.

Parameters:

[in]	*This*	Pointer to EFI_DNS6_PROTOCOL instance.

[out] *DnsModeData* Pointer to the caller-allocated storage for the EFI_DNS6_MODE_DATA structure.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_STARTED	When DnsConfigData is queried, no configuration data is available because this instance has not been configured.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL or DnsModeData is NULL.

References _DNS_DRIVER_DATA::Dns6CacheList, _DNS_INSTANCE::Dns6CfgData, Dns6CopyConfigure(), DNS6_SERVER_IP::Dns6ServerIp, _DNS_DRIVER_DATA::Dns6ServerList, DNS_INSTANCE_FROM_THIS_PROTOCOL6, DNS_STATE_UNCONFIGED, DNS6_CACHE::DnsCache, mDriverData, and _DNS_INSTANCE::State.

```
EFI_STATUS EFIAPI Dns6HostNameToIp ( IN EFI_DNS6_PROTOCOL *          This,
                                     IN CHAR16 *                     HostName,
                                     IN EFI_DNS6_COMPLETION_TOKEN *  Token
                                   )
```

The function is used to translate the host name to host IP address. A type AAAA query is used to get the one or more IPv6 addresses for this host.

Parameters:

[in] *This*	Pointer to EFI_DNS6_PROTOCOL instance.
[in] *HostName*	Pointer to caller-supplied buffer containing Host name to be translated. This buffer contains 16 bit characters but these are translated to ASCII for use with DNSv4 server and there is no requirement for driver to support non-ASCII Unicode characters.
[in] *Token*	Pointer to the caller-allocated completion token to return at the completion of the process to translate host name to host address.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL. Token is NULL. Token.Event is.NULL HostName is NULL
EFI_NO_MAPPING	There's no source address is available for use.
EFI_NOT_STARTED	This instance has not been started.

References ConstructDNSQueryIp(), _DNS_DRIVER_DATA::Dns6CacheList, _DNS_INSTANCE::Dns6CfgData, _DNS_INSTANCE::Dns6TxTokens, DNS_DEFAULT_RETRY, DNS_DEFAULT_TIMEOUT, DNS_INSTANCE_FROM_THIS_PROTOCOL6, DNS_STATE_CONFIGED, DNS_TYPE_AAAA, DNS6_CACHE::DnsCache, DoDnsQuery(), _DNS_INSTANCE::MaxRetry, mDriverData, DNS6_TOKEN_ENTRY::PacketToLive, DNS6_TOKEN_ENTRY::QueryHostName, _DNS_INSTANCE::State, and DNS6_TOKEN_ENTRY::Token.

```
EFI_STATUS EFIAPI Dns6IpToHostName ( IN EFI_DNS6_PROTOCOL *          This,
                                     IN EFI_IPv6_ADDRESS             IpAddress,
                                     IN EFI_DNS6_COMPLETION_TOKEN *  Token
                                   )
```

The function is used to translate the host address to host name. A type PTR query is used to get the primary name of the host.

Parameters:

[in]	*This*	Pointer to EFI_DNS6_PROTOCOL instance.
[in]	*IpAddress*	IP address.
[in]	*Token*	Pointer to the caller-allocated completion used token to translate host address to host name.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_INVALID_PARAMETER	This is NULL. Token is NULL. Token.Event is NULL. IpAddress is not valid IP address.
EFI_NO_MAPPING	There's no source address is available for use.
EFI_NOT_STARTED	This instance has not been started.
EFI_UNSUPPORTED	This function is not supported.

EFI_STATUS EFIAPI Dns6Poll (IN EFI_DNS6_PROTOCOL * *This*)

This function can be used by network drivers and applications to increase the rate that data packets are moved between the communications device and the transmit and receive queues. In some systems, the periodic timer event in the managed network driver may not poll the underlying communications device fast enough to transmit and/or receive all data packets without missing incoming packets or dropping outgoing packets.

Parameters:

[in]	*This*	Pointer to EFI_DNS6_PROTOCOL instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This EFI DNS Protocol instance has not been started.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_TIMEOUT	Data was dropped out of the transmit and/or receive queue. Consider increasing the polling rate.

References DNS_INSTANCE_FROM_THIS_PROTOCOL6, DNS_STATE_DESTROY, DNS_STATE_UNCONFIGED, _DNS_INSTANCE::State, and _DNS_INSTANCE::UdpIo.

```
EFI_STATUS EFIAPI Dns6UpdateDnsCache ( IN EFI_DNS6_PROTOCOL *      This,
                                       IN BOOLEAN                  DeleteFlag,
                                       IN BOOLEAN                  Override,
                                       IN EFI_DNS6_CACHE_ENTRY     DnsCacheEntry
                                     )
```

This function is used to add/delete/modify DNS cache entry. DNS cache can be normally dynamically updated after the DNS resolve succeeds. This function provided capability to manually add/delete/modify the DNS cache.

Parameters:

[in]	*This*	Pointer to EFI_DNS6_PROTOCOL instance.
[in]	*DeleteFlag*	

	If FALSE, this function is to add one entry to the DNS Cache. If TRUE, this function will delete matching DNS Cache entry.
[in] *Override*	If TRUE, the matching DNS cache entry will be overwritten with the supplied parameter. If FALSE, EFI_ACCESS_DENIED will be returned if the entry to be added is already exists.
[in] *DnsCacheEntry*	Pointer to DNS Cache entry.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	This is NULL. DnsCacheEntry.HostName is NULL. DnsCacheEntry.IpAddress is NULL. DnsCacheEntry.Timeout is zero.
EFI_ACCESS_DENIED	The DNS cache entry already exists and Override is not TRUE.

References _DNS_DRIVER_DATA::Dns6CacheList, mDriverData, and UpdateDns6Cache().

Variable Documentation

EFI_DNS4_PROTOCOL mDns4Protocol

Initial value:

```
{
 Dns4GetModeData,
 Dns4Configure,
 Dns4HostNameToIp,
 Dns4IpToHostName,
 Dns4GeneralLookUp,
 Dns4UpdateDnsCache,
 Dns4Poll,
 Dns4Cancel
}
```

Referenced by DnsCreateInstance().

EFI_DNS6_PROTOCOL mDns6Protocol

Initial value:

```
{
 Dns6GetModeData,
 Dns6Configure,
 Dns6HostNameToIp,
 Dns6IpToHostName,
 Dns6GeneralLookUp,
 Dns6UpdateDnsCache,
 Dns6Poll,
 Dns6Cancel
}
```

Referenced by DnsCreateInstance().

NetworkPkg/HttpBootDxe/HttpBootClient.c File Reference

Functions

EFI_STATUS HttpBootUpdateDevicePath (IN HTTP_BOOT_PRIVATE_DATA *Private)
EFI_STATUS HttpBootExtractUriInfo (IN HTTP_BOOT_PRIVATE_DATA *Private)
EFI_STATUS HttpBootDiscoverBootInfo (IN OUT HTTP_BOOT_PRIVATE_DATA *Private)
EFI_STATUS HttpBootCreateHttpIo (IN HTTP_BOOT_PRIVATE_DATA *Private)
EFI_STATUS HttpBootGetFileFromCache (IN HTTP_BOOT_PRIVATE_DATA *Private, IN CHAR16 *Uri, IN OUT UINTN *BufferSize, OUT UINT8 *Buffer)
VOID HttpBootFreeCache (IN HTTP_BOOT_CACHE_CONTENT *Cache)
VOID HttpBootFreeCacheList (IN HTTP_BOOT_PRIVATE_DATA *Private)
EFI_STATUS EFIAPI HttpBootGetBootFileCallback (IN HTTP_BODY_PARSE_EVENT EventType, IN CHAR8 *Data, IN UINTN Length, IN VOID *Context)
EFI_STATUS HttpBootGetBootFile (IN HTTP_BOOT_PRIVATE_DATA *Private, IN BOOLEAN HeaderOnly, IN OUT UINTN *BufferSize, OUT UINT8 *Buffer)

Detailed Description

Implementation of the boot file download function.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS HttpBootCreateHttpIo (IN HTTP_BOOT_PRIVATE_DATA * *Private*)

Create a HttpIo instance for the file download.

Parameters:

[in]	*Private*	The pointer to the driver's private data.

Return values:

EFI_SUCCESS	Successfully created.
Others	Failed to create HttpIo.

References HTTP_IO_CONFIG_DATA::Config4, HTTP_BOOT_REQUEST_TIMEOUT, HttpIoCreateIo(), HTTP4_IO_CONFIG_DATA::HttpVersion, HTTP4_IO_CONFIG_DATA::LocalIp, HTTP4_IO_CONFIG_DATA::RequestTimeOut, and HTTP4_IO_CONFIG_DATA::SubnetMask.

Referenced by HttpBootLoadFile().

EFI_STATUS HttpBootDiscoverBootInfo (IN OUT HTTP_BOOT_PRIVATE_DATA * *Private*)
Discover all the boot information for boot file.

Parameters:

`[in,out]` *Private* The pointer to the driver's private data.

Return values:

EFI_SUCCESS Successfully obtained all the boot information .
Others Failed to retrieve the boot information.

References HttpBootDhcp(), and HttpBootExtractUriInfo().

Referenced by HttpBootLoadFile().

EFI_STATUS HttpBootExtractUriInfo (IN HTTP_BOOT_PRIVATE_DATA * *Private*)
Parse the boot file URI information from the selected Dhcp4 offer packet.

Parameters:

`[in]` *Private* The pointer to the driver's private data.

Return values:

EFI_SUCCESS Successfully parsed out all the boot information.
Others Failed to parse out the boot information.

References HTTP_BOOT_DHCP4_TAG_INDEX_BOOTFILE, HTTP_BOOT_DHCP4_TAG_INDEX_DNS_SERVER, HTTP_BOOT_OFFER_MAX_NUM, HttpBootRegisterIp4Dns(), HttpBootUpdateDevicePath(), HttpOfferTypeDhcpDns, HttpOfferTypeDhcpIpUri, HttpOfferTypeDhcpNameUriDns, HttpOfferTypeMax, HTTP_BOOT_DHCP4_PACKET_CACHE::OfferType, HTTP_BOOT_DHCP4_PACKET_CACHE::OptList, and HTTP_BOOT_DHCP4_PACKET_CACHE::UriParser.

Referenced by HttpBootDiscoverBootInfo().

VOID HttpBootFreeCache (IN HTTP_BOOT_CACHE_CONTENT * *Cache*)
Release all the resource of a cache item.

Parameters:

`[in]` *Cache* The pointer to the cache item.

References HTTP_BOOT_ENTITY_DATA::Block, and HTTP_BOOT_ENTITY_DATA::Link.

Referenced by HttpBootFreeCacheList(), and HttpBootGetBootFile().

VOID HttpBootFreeCacheList (IN HTTP_BOOT_PRIVATE_DATA * *Private*)
Clean up all cached data.

Parameters:

`[in]` *Private* The pointer to the driver's private

data.

References HttpBootFreeCache(), and HTTP_BOOT_CACHE_CONTENT::Link.

Referenced by HttpBootIp4DxeDriverBindingStop().

```
EFI_STATUS HttpBootGetBootFile ( IN HTTP_BOOT_PRIVATE_DATA *   Private,
                                 IN BOOLEAN                    HeaderOnly,
                                 IN OUT UINTN *                BufferSize,
                                 OUT UINT8 *                   Buffer
                               )
```

This function download the boot file by using UEFI HTTP protocol.

Parameters:

[in]	*Private*	The pointer to the driver's private data.
[in]	*HeaderOnly*	Only request the response header, it could save a lot of time if the caller only want to know the size of the requested file.
[in,out]	*BufferSize*	On input the size of Buffer in bytes. On output with a return code of EFI_SUCCESS, the amount of data transferred to Buffer. On output with a return code of EFI_BUFFER_TOO_SMALL, the size of Buffer required to retrieve the requested file.
[out]	*Buffer*	The memory buffer to transfer the file to. IF Buffer is NULL, then the size of the requested file is returned in BufferSize.

Return values:

EFI_SUCCESS	The file was loaded.
EFI_INVALID_PARAMETER	BufferSize is NULL or Buffer Size is not NULL but Buffer is NULL.
EFI_OUT_OF_RESOURCES	Could not allocate needed resources
EFI_BUFFER_TOO_SMALL	The BufferSize is too small to read the current directory entry. BufferSize has been updated with the size needed to complete the request.
Others	Unexpected error happened.

References HTTP_BOOT_CALLBACK_DATA::Block, HTTP_IO_RESOPNSE_DATA::Body, HTTP_IO_RESOPNSE_DATA::BodyLength, HTTP_BOOT_CALLBACK_DATA::Buffer, HTTP_BOOT_CALLBACK_DATA::BufferSize, HTTP_BOOT_CALLBACK_DATA::Cache, HTTP_BOOT_CALLBACK_DATA::CopyedSize, HTTP_BOOT_CACHE_CONTENT::EntityDataList, HTTP_BOOT_CACHE_CONTENT::EntityLength, HTTP_IO_RESOPNSE_DATA::HeaderCount, HTTP_IO_HEADER::HeaderCount, HTTP_IO_RESOPNSE_DATA::Headers, HTTP_IO_HEADER::Headers, HTTP_BOOT_BLOCK_SIZE, HTTP_FIELD_NAME_ACCEPT, HTTP_FIELD_NAME_HOST, HTTP_FIELD_NAME_USER_AGENT, HTTP_USER_AGENT_EFI_HTTP_BOOT, HttpBootCreateHeader(), HttpBootFreeCache(), HttpBootFreeHeader(), HttpBootGetBootFileCallback(), HttpBootGetFileFromCache(), HttpBootSetHeader(), HttpIoRecvResponse(), HttpIoSendRequest(), HTTP_BOOT_CACHE_CONTENT::Link, HTTP_BOOT_CALLBACK_DATA::NewBlock, HTTP_BOOT_CACHE_CONTENT::RequestData, HTTP_IO_RESOPNSE_DATA::Response, and HTTP_BOOT_CACHE_CONTENT::ResponseData.

Referenced by HttpBootLoadFile().

```
EFI_STATUS EFIAPI HttpBootGetBootFileCallback ( IN HTTP_BODY_PARSE_EVENT  EventType,
                                                IN CHAR8 *                Data,
                                                IN UINTN                  Length,
                                                IN VOID *                 Context
                                              )
```

A callback function to intercept events during message parser.

This function will be invoked during HttpParseMessageBody() with various events type. An error return status of the callback function will cause the HttpParseMessageBody() aborted.

Parameters:

[in]	*EventType*	Event type of this callback call.
[in]	*Data*	A pointer to data buffer.
[in]	*Length*	Length in bytes of the Data.
[in]	*Context*	Callback context set by HttpInitMsgParser().

Return values:

EFI_SUCCESS	Continue to parser the message body.
Others	Abort the parse.

References HTTP_BOOT_CALLBACK_DATA::Block, HTTP_BOOT_ENTITY_DATA::Block, HTTP_BOOT_CALLBACK_DATA::Buffer, HTTP_BOOT_CALLBACK_DATA::BufferSize, HTTP_BOOT_CALLBACK_DATA::Cache, HTTP_BOOT_CALLBACK_DATA::CopyedSize, HTTP_BOOT_ENTITY_DATA::DataLength, HTTP_BOOT_ENTITY_DATA::DataStart, HTTP_BOOT_CACHE_CONTENT::EntityDataList, HTTP_BOOT_ENTITY_DATA::Link, and HTTP_BOOT_CALLBACK_DATA::NewBlock.

Referenced by HttpBootGetBootFile().

```
EFI_STATUS HttpBootGetFileFromCache ( IN HTTP_BOOT_PRIVATE_DATA *  Private,
                                      IN CHAR16 *                 Uri,
                                      IN OUT UINTN *              BufferSize,
                                      OUT UINT8 *                 Buffer
                                    )
```

Get the file content from cached data.

Parameters:

[in]	*Private*	The pointer to the driver's private data.
[in]	*Uri*	Uri of the file to be retrieved from cache.
[in,out]	*BufferSize*	On input the size of Buffer in bytes. On output with a return code of EFI_SUCCESS, the amount of data transferred to Buffer. On output with a return code of EFI_BUFFER_TOO_SMALL, the size of Buffer required to retrieve the requested file.
[out]	*Buffer*	The memory buffer to transfer the file to. IF Buffer is NULL, then the size of the requested file is returned in BufferSize.

Return values:

EFI_SUCCESS	Successfully created.
Others	Failed to create HttpIo.

References HTTP_BOOT_ENTITY_DATA::DataLength, HTTP_BOOT_ENTITY_DATA::DataStart, HTTP_BOOT_CACHE_CONTENT::EntityDataList, HTTP_BOOT_CACHE_CONTENT::EntityLength, and HTTP_BOOT_CACHE_CONTENT::RequestData.

Referenced by HttpBootGetBootFile().

EFI_STATUS HttpBootUpdateDevicePath (IN HTTP_BOOT_PRIVATE_DATA * *Private*)

Update the IP and URL device path node to include the boot resource information.

Parameters:

[in]	*Private*	The pointer to the driver's private data.

Return values:

EFI_SUCCESS	Device patch successfully updated.
EFI_OUT_OF_RESOURCES	Could not allocate needed resources.
Others	Unexpected error happened.

Referenced by HttpBootExtractUriInfo().

NetworkPkg/HttpBootDxe/HttpBootClient.h File Reference

Data Structures

struct HTTP_BOOT_ENTITY_DATA
struct HTTP_BOOT_CACHE_CONTENT
struct HTTP_BOOT_CALLBACK_DAT

Defines

"memItemRight" valign="bottom">HTTP_BOOT_REQUEST_TIMEOUT 5000
#define HTTP_BOOT_BLOCK_SIZE 150
#define HTTP_FIELD_NAME_USER_AG
#define HTTP_FIELD_NAME_HOST "H
#define HTTP_FIELD_NAME_ACCEPT
#define HTTP_USER_AGENT_EFI_HTTP

Functions

EFI_STATUS HttpBootDiscoverBootInfo (IN OU
HTTP_BOOT_PRIVATE_DATA *
EFI_STATUS HttpBootCreateHttpIo (IN HTTP_B
*Private)
EFI_STATUS HttpBootGetBootFile (IN HTTP_B
*Private, IN BOOLEAN HeaderOn
*BufferSize, OUT UINT8 *Buffer)
VOID HttpBootFreeCacheList (IN HTTP_
*Private)

Detailed Description

Declaration of the boot file download function.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by HttpBootGetBootFile().

#define HTTP_BOOT_REQUEST_TIMEOUT 5000

Referenced by HttpBootCreateHttpIo().

#define HTTP_FIELD_NAME_ACCEPT "Accept"

Referenced by HttpBootGetBootFile().

#define HTTP_FIELD_NAME_HOST "Host"

Referenced by HttpBootGetBootFile().

#define HTTP_FIELD_NAME_USER_AGENT "User-Agent"

Referenced by HttpBootGetBootFile().

#define HTTP_USER_AGENT_EFI_HTTP_BOOT "UefiHttpBoot/1.0"

Referenced by HttpBootGetBootFile().

Function Documentation

EFI_STATUS HttpBootCreateHttpIo (IN HTTP_BOOT_PRIVATE_DATA * *Private*)
Create a HttpIo instance for the file download.

Parameters:

[in]	*Private*	The pointer to the driver's private data.

Return values:

EFI_SUCCESS	Successfully created.
Others	Failed to create HttpIo.

References HTTP_IO_CONFIG_DATA::Config4, HTTP_BOOT_REQUEST_TIMEOUT, HttpIoCreateIo(), HTTP4_IO_CONFIG_DATA::HttpVersion, HTTP4_IO_CONFIG_DATA::LocalIp, HTTP4_IO_CONFIG_DATA::RequestTimeOut, and HTTP4_IO_CONFIG_DATA::SubnetMask.

Referenced by HttpBootLoadFile().

EFI_STATUS HttpBootDiscoverBootInfo (IN OUT HTTP_BOOT_PRIVATE_DATA * *Private*)
Discover all the boot information for boot file.

Parameters:

[in,out]	*Private*	The pointer to the driver's private data.

Return values:

EFI_SUCCESS	Successfully obtained all the boot information .
Others	Failed to retrieve the boot information.

References HttpBootDhcp(), and HttpBootExtractUriInfo().

Referenced by HttpBootLoadFile().

VOID HttpBootFreeCacheList (IN HTTP_BOOT_PRIVATE_DATA * *Private*)

Clean up all cached data.

Parameters:

[in]	*Private*	The pointer to the driver's private data.

References HttpBootFreeCache(), and HTTP_BOOT_CACHE_CONTENT::Link.

Referenced by HttpBootIp4DxeDriverBindingStop().

EFI_STATUS HttpBootGetBootFile (IN HTTP_BOOT_PRIVATE_DATA * *Private*,
IN BOOLEAN *HeaderOnly*,
IN OUT UINTN * *BufferSize*,
OUT UINT8 * *Buffer*
)

This function download the boot file by using UEFI HTTP protocol.

Parameters:

[in]	*Private*	The pointer to the driver's private data.
[in]	*HeaderOnly*	Only request the response header, it could save a lot of time if the caller only want to know the size of the requested file.
[in,out]	*BufferSize*	On input the size of Buffer in bytes. On output with a return code of EFI_SUCCESS, the amount of data transferred to Buffer. On output with a return code of EFI_BUFFER_TOO_SMALL, the size of Buffer required to retrieve the requested file.
[out]	*Buffer*	The memory buffer to transfer the file to. IF Buffer is NULL, then the size of the requested file is returned in BufferSize.

Return values:

EFI_SUCCESS	The file was loaded.
EFI_INVALID_PARAMETER	BufferSize is NULL or Buffer Size is not NULL but Buffer is NULL.
EFI_OUT_OF_RESOURCES	Could not allocate needed resources
EFI_BUFFER_TOO_SMALL	The BufferSize is too small to read the current directory entry. BufferSize has been updated with the size needed to complete the request.
Others	Unexpected error happened.

References HTTP_BOOT_CALLBACK_DATA::Block, HTTP_IO_RESOPNSE_DATA::Body, HTTP_IO_RESOPNSE_DATA::BodyLength, HTTP_BOOT_CALLBACK_DATA::Buffer, HTTP_BOOT_CALLBACK_DATA::BufferSize, HTTP_BOOT_CALLBACK_DATA::Cache, HTTP_BOOT_CALLBACK_DATA::CopyedSize, HTTP_BOOT_CACHE_CONTENT::EntityDataList, HTTP_BOOT_CACHE_CONTENT::EntityLength, HTTP_IO_RESOPNSE_DATA::HeaderCount,

HTTP_IO_HEADER::HeaderCount, HTTP_IO_RESOPNSE_DATA::Headers, HTTP_IO_HEADER::Headers, HTTP_BOOT_BLOCK_SIZE, HTTP_FIELD_NAME_ACCEPT, HTTP_FIELD_NAME_HOST, HTTP_FIELD_NAME_USER_AGENT, HTTP_USER_AGENT_EFI_HTTP_BOOT, HttpBootCreateHeader(), HttpBootFreeCache(), HttpBootFreeHeader(), HttpBootGetBootFileCallback(), HttpBootGetFileFromCache(), HttpBootSetHeader(), HttpIoRecvResponse(), HttpIoSendRequest(), HTTP_BOOT_CACHE_CONTENT::Link, HTTP_BOOT_CALLBACK_DATA::NewBlock, HTTP_BOOT_CACHE_CONTENT::RequestData, HTTP_IO_RESOPNSE_DATA::Response, and HTTP_BOOT_CACHE_CONTENT::ResponseData.

Referenced by HttpBootLoadFile().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/HttpBootDxe/HttpBootComponentName.c File Reference

Functions

EFI_STATUS EFIAPI HttpBootDxeComponentNameGetDriverName (IN EFI_COMPONENT_NAME2_PROTOCOL *This, IN CHAR8 *Language, OUT CHAR16 **DriverName)

EFI_STATUS EFIAPI HttpBootDxeComponentNameGetControllerName (IN EFI_COMPONENT_NAME2_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE ChildHandle, IN CHAR8 *Language, OUT CHAR16 **ControllerName)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gHttpBootDxeComponentName

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gHttpBootDxeComponentName2

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mHttpBootDxeDriverNameTable []

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mHttpBootDxeControllerNameTable []

Detailed Description

Implementation of EFI_COMPONENT_NAME_PROTOCOL and EFI_COMPONENT_NAME2_PROTOCOL protocol.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI HttpBootDxeComponentNameGetControllerName	(	IN EFI_COMPONENT_NAME2_PROTOCOL *	*This*,
		IN EFI_HANDLE	*ControllerHandle*
		IN EFI_HANDLE	*ChildHandle*,
		IN CHAR8 *	*Language*,

OUT CHAR16 ** *ControllerName*
)

Retrieves a Unicode string that is the user readable name of the controller that is being managed by an EFI Driver.

Parameters:

This	A pointer to the EFI_COMPONENT_NAME_PROTOCOL instance.
ControllerHandle	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
ChildHandle	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
Language	A pointer to a three character ISO 639-2 language identifier. This is the language of the controller name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer.
ControllerName	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language, from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gHttpBootDxeComponentName2, and HttpBootGetNicByIp4Children().

EFI_STATUS EFIAPI
HttpBootDxeComponentNameGetDriverName
(IN EFI_COMPONENT_NAME2_PROTOCOL * *This*,
IN CHAR8 * *Language*,
OUT CHAR16 ** *DriverName*
)

Retrieves a Unicode string that is the user-readable name of the EFI Driver.

Parameters:

This	A pointer to the EFI_COMPONENT_NAME_PROTOCOL instance.
Language	A pointer to a three-character ISO 639-2 language identifier. This is the language of the driver name that that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer.
DriverName	

A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by This in the language specified by Language.

Return values:

EFI_SUCCESS	The Unicode string for the Driver specified by This and the language specified by Language was returned in DriverName.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	DriverName is NULL.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gHttpBootDxeComponentName2, and mHttpBootDxeDriverNameTable.

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gHttpBootDxeComponentName

Initial value:

```
{
 (EFI_COMPONENT_NAME_GET_DRIVER_NAME)     HttpBootDxeComponentNameGetDriverName,
 (EFI_COMPONENT_NAME_GET_CONTROLLER_NAME)HttpBootDxeComponentNameGetControllerName,
 "eng"
}
```

Component Name Protocol instance

Referenced by HttpBootDxeDriverEntryPoint().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gHttpBootDxeComponentName2

Initial value:

```
{
 HttpBootDxeComponentNameGetDriverName,
 HttpBootDxeComponentNameGetControllerName,
 "en"
}
```

Component Name 2 Protocol instance

Referenced by HttpBootDxeComponentNameGetControllerName(), HttpBootDxeComponentNameGetDriverName(), and HttpBootDxeDriverEntryPoint().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mHttpBootDxeControllerNameTable[]

Initial value:

```
{
 { "eng;en", (CHAR16 *)L"UEFI Http Boot Controller" },
 { NULL, NULL }
}
```

Table of controller names

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mHttpBootDxeDriverNameTable[]

Initial value:

```
 {
  { "eng;en", (CHAR16 *)L"UEFI HTTP Boot Driver" },
  { NULL, NULL }
}
```

Table of driver names

Referenced by HttpBootDxeComponentNameGetDriverName().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/HttpBootDxe/HttpBootComponentName.h File Reference

Functions

EFI_STATUS EFIAPI HttpBootDxeComponentNameGetDriverName (IN EFI_COMPONENT_NAME2_PROTOCOL *This, IN CHAR8 *Language, OUT CHAR16 **DriverName)

EFI_STATUS EFIAPI HttpBootDxeComponentNameGetControllerName (IN EFI_COMPONENT_NAME2_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE ChildHandle, IN CHAR8 *Language, OUT CHAR16 **ControllerName)

Detailed Description

Declaration of HTTP boot driver's EFI_COMPONENT_NAME_PROTOCOL and EFI_COMPONENT_NAME2_PROTOCOL function.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI
HttpBootDxeComponentNameGetControllerName (
    IN EFI_COMPONENT_NAME2_PROTOCOL *  This,
    IN EFI_HANDLE                      ControllerHandle,
    IN EFI_HANDLE                      ChildHandle,
    IN CHAR8 *                         Language,
    OUT CHAR16 **                      ControllerName
)
```

Retrieves a Unicode string that is the user readable name of the controller that is being managed by an EFI Driver.

Parameters:

This	A pointer to the EFI_COMPONENT_NAME_PROTOCOL instance.
ControllerHandle	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
ChildHandle	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will

not be NULL for a bus driver that wishes to retrieve the name of a child controller.

Language A pointer to a three character ISO 639-2 language identifier. This is the language of the controller name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer.

ControllerName A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language, from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gHttpBootDxeComponentName2, and HttpBootGetNicByIp4Children().

```
EFI_STATUS EFIAPI
HttpBootDxeComponentNameGetDriverName (
  IN EFI_COMPONENT_NAME2_PROTOCOL *   This,
  IN CHAR8 *                          Language,
  OUT CHAR16 **                       DriverName
)
```

Retrieves a Unicode string that is the user-readable name of the EFI Driver.

Parameters:

This A pointer to the EFI_COMPONENT_NAME_PROTOCOL instance.

Language A pointer to a three-character ISO 639-2 language identifier. This is the language of the driver name that that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer.

DriverName A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by This in the language specified by Language.

Return values:

EFI_SUCCESS	The Unicode string for the Driver specified by This and the language specified by Language was returned in DriverName.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	DriverName is NULL.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gHttpBootDxeComponentName2, and mHttpBootDxeDriverNameTable.

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/HttpBootDxe/HttpBootDhcp4.c File Reference

Functions

UINT32 HttpBootBuildDhcp4Options (IN HTTP_BOOT_PRIVATE_DATA *Private, OUT EFI_DHCP4_PACKET_OPTION **OptList, IN UINT8 *Buffer)

EFI_DHCP4_PACKET_OPTION * HttpBootParseDhcp4Options (IN UINT8 *Buffer, IN UINT32 Length, IN UINT8 OptTag)

VOID HttpBootCacheDhcp4Packet (IN EFI_DHCP4_PACKET *Dst, IN EFI_DHCP4_PACKET *Src)

EFI_STATUS HttpBootParseDhcp4Packet (IN HTTP_BOOT_DHCP4_PACKET_CACHE *Cache4)

VOID HttpBootCacheDhcp4Offer (IN HTTP_BOOT_PRIVATE_DATA *Private, IN EFI_DHCP4_PACKET *RcvdOffer)

VOID HttpBootSelectDhcp4Offer (IN HTTP_BOOT_PRIVATE_DATA *Private)

EFI_STATUS EFIAPI HttpBootDhcp4CallBack (IN EFI_DHCP4_PROTOCOL *This, IN VOID *Context, IN EFI_DHCP4_STATE CurrentState, IN EFI_DHCP4_EVENT Dhcp4Event, IN EFI_DHCP4_PACKET *Packet, OUT EFI_DHCP4_PACKET **NewPacket)

EFI_STATUS HttpBootRegisterIp4Gateway (IN HTTP_BOOT_PRIVATE_DATA *Private)

EFI_STATUS HttpBootRegisterIp4Dns (IN HTTP_BOOT_PRIVATE_DATA *Private, IN UINTN DataLength, IN VOID *DnsServerData)

EFI_STATUS HttpBootSetIpPolicy (IN HTTP_BOOT_PRIVATE_DATA *Private)

EFI_STATUS HttpBootDhcp4Dora (IN HTTP_BOOT_PRIVATE_DATA *Private)

Variables

UINT8 mInterestedDhcp4Tags [HTTP_BOOT_DHCP4_TAG_INDEX_MAX]

UINT32 mHttpDhcpTimeout [4] = {4, 8, 16, 32}

Detailed Description

Functions implementation related with DHCPv4 for HTTP boot driver.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
UINT32 HttpBootBuildDhcp4Options ( IN HTTP_BOOT_PRIVATE_DATA *        Private,
                                   OUT EFI_DHCP4_PACKET_OPTION **     OptList,
                                   IN UINT8 *                         Buffer
                                 )
```

Build the options buffer for the DHCPv4 request packet.

Parameters:

[in]	*Private*	Pointer to HTTP boot driver private data.
[out]	*OptList*	Pointer to the option pointer array.
[in]	*Buffer*	Pointer to the buffer to contain the option list.

Returns:

Index The count of the built-in options.

References HTTP_BOOT_DHCP4_OPTION_ENTRY::Arch, HTTP_BOOT_DHCP4_OPTION_CLID::ArchitectureType, HTTP_BOOT_DHCP4_OPTION_ENTRY::Clid, DEFAULT_CLASS_ID_DATA, DEFAULT_UNDI_MAJOR, DEFAULT_UNDI_MINOR, DEFAULT_UNDI_TYPE, EFI_HTTP_BOOT_CLIENT_SYSTEM_ARCHITECTURE, GET_NEXT_DHCP_OPTION, HTTP_BOOT_DHCP4_OPTION_UUID::Guid, HTTP_BOOT_DHCP4_TAG_ARCH, HTTP_BOOT_DHCP4_TAG_BOOTFILE, HTTP_BOOT_DHCP4_TAG_BOOTFILE_LEN, HTTP_BOOT_DHCP4_TAG_BROADCAST, HTTP_BOOT_DHCP4_TAG_CLASS_ID, HTTP_BOOT_DHCP4_TAG_DNS_SERVER, HTTP_BOOT_DHCP4_TAG_DOMAINNAME, HTTP_BOOT_DHCP4_TAG_EMTU, HTTP_BOOT_DHCP4_TAG_EXTEND_PATH, HTTP_BOOT_DHCP4_TAG_HOSTNAME, HTTP_BOOT_DHCP4_TAG_LEASE, HTTP_BOOT_DHCP4_TAG_NAME_SERVER, HTTP_BOOT_DHCP4_TAG_NETMASK, HTTP_BOOT_DHCP4_TAG_NIS_DOMAIN, HTTP_BOOT_DHCP4_TAG_NIS_SERVER, HTTP_BOOT_DHCP4_TAG_NTP_SERVER, HTTP_BOOT_DHCP4_TAG_PARA_LIST, HTTP_BOOT_DHCP4_TAG_REQUEST_IP, HTTP_BOOT_DHCP4_TAG_ROOTPATH, HTTP_BOOT_DHCP4_TAG_ROUTER, HTTP_BOOT_DHCP4_TAG_SERVER_ID, HTTP_BOOT_DHCP4_TAG_T1, HTTP_BOOT_DHCP4_TAG_T2, HTTP_BOOT_DHCP4_TAG_TIME_OFFSET, HTTP_BOOT_DHCP4_TAG_TIME_SERVER, HTTP_BOOT_DHCP4_TAG_TTL, HTTP_BOOT_DHCP4_TAG_UNDI, HTTP_BOOT_DHCP4_TAG_UUID, HTTP_BOOT_DHCP4_TAG_VENDOR, HttpBootUintnToAscDecWithFormat(), HTTP_BOOT_DHCP4_OPTION_CLID::InterfaceName, HTTP_BOOT_DHCP4_OPTION_UNDI::MajorVer, HTTP_BOOT_DHCP4_OPTION_UNDI::MinorVer, HTTP_BOOT_DHCP4_OPTION_ENTRY::Para, HTTP_BOOT_DHCP4_OPTION_PARA::ParaList, HTTP_BOOT_DHCP4_OPTION_ARCH::Type, HTTP_BOOT_DHCP4_OPTION_UNDI::Type, HTTP_BOOT_DHCP4_OPTION_UUID::Type, HTTP_BOOT_DHCP4_OPTION_ENTRY::Undi, HTTP_BOOT_DHCP4_OPTION_CLID::UndiMajor, HTTP_BOOT_DHCP4_OPTION_CLID::UndiMinor, and HTTP_BOOT_DHCP4_OPTION_ENTRY::Uuid.

Referenced by HttpBootDhcp4Dora().

```
VOID HttpBootCacheDhcp4Offer ( IN HTTP_BOOT_PRIVATE_DATA *   Private,
                               IN EFI_DHCP4_PACKET *         RcvdOffer
                             )
```

Cache all the received DHCPv4 offers, and set OfferIndex and OfferCount.

Parameters:

[in]	*Private*	Pointer to HTTP boot driver private data.
[in]	*RcvdOffer*	Pointer to the received offer packet.

References HTTP_BOOT_OFFER_MAX_NUM, HttpBootCacheDhcp4Packet(), HttpBootParseDhcp4Packet(), HttpOfferTypeMax, HTTP_BOOT_DHCP4_PACKET::Offer, HTTP_BOOT_DHCP4_PACKET_CACHE::OfferType, and HTTP_BOOT_DHCP4_PACKET_CACHE::Packet.

Referenced by HttpBootDhcp4CallBack().

```
VOID HttpBootCacheDhcp4Packet ( IN EFI_DHCP4_PACKET *  Dst,
                                IN EFI_DHCP4_PACKET *  Src
                              )
```

Cache the DHCPv4 packet.

Parameters:

[in]	*Dst*	Pointer to the cache buffer for DHCPv4 packet.
[in]	*Src*	Pointer to the DHCPv4 packet to be cached.

Referenced by HttpBootCacheDhcp4Offer().

```
EFI_STATUS EFIAPI HttpBootDhcp4CallBack ( IN EFI_DHCP4_PROTOCOL *   This,
                                          IN VOID *                 Context,
                                          IN EFI_DHCP4_STATE        CurrentState,
                                          IN EFI_DHCP4_EVENT        Dhcp4Event,
                                          IN EFI_DHCP4_PACKET *     Packet,
                                          OUT EFI_DHCP4_PACKET **   NewPacket
                                        )
```

EFI_DHCP4_CALLBACK is provided by the consumer of the EFI DHCPv4 Protocol driver to intercept events that occurred in the configuration process.

Parameters:

[in]	*This*	Pointer to the EFI DHCPv4 Protocol.
[in]	*Context*	Pointer to the context set by EFI_DHCP4_PROTOCOL.Configure().
[in]	*CurrentState*	The current operational state of the EFI DHCPv4 Protocol driver.
[in]	*Dhcp4Event*	The event that occurs in the current state, which usually means a state transition.
[in]	*Packet*	The DHCPv4 packet that is going to be sent or already received.
[out]	*NewPacket*	The packet that is used to replace the above Packet.

Return values:

EFI_SUCCESS	Tells the EFI DHCPv4 Protocol driver to continue the DHCP process.
EFI_NOT_READY	Only used in the Dhcp4Selecting state. The EFI DHCPv4 Protocol driver will continue to wait for more DHCPOFFER packets until the retry timeout expires.
EFI_ABORTED	Tells the EFI DHCPv4 Protocol driver to abort the current process and return to the Dhcp4Init or Dhcp4InitReboot state.

References HTTP_BOOT_DHCP_PACKET_CACHE::Dhcp4, GET_OPTION_BUFFER_LEN,

HTTP_BOOT_DHCP4_PACKET_MAX_SIZE, HTTP_BOOT_DHCP4_TAG_MAXMSG, HTTP_BOOT_OFFER_MAX_NUM, HttpBootCacheDhcp4Offer(), HttpBootParseDhcp4Options(), HttpBootSelectDhcp4Offer(), HTTP_BOOT_DHCP4_PACKET::Offer, _HTTP_BOOT_PRIVATE_DATA::OfferBuffer, _HTTP_BOOT_PRIVATE_DATA::OfferNum, HTTP_BOOT_DHCP4_PACKET_CACHE::Packet, and _HTTP_BOOT_PRIVATE_DATA::SelectIndex.

Referenced by HttpBootDhcp4Dora().

EFI_STATUS HttpBootDhcp4Dora (IN HTTP_BOOT_PRIVATE_DATA * *Private*)

Start the D.O.R.A DHCPv4 process to acquire the IPv4 address and other Http boot information.

Parameters:

[in] *Private* Pointer to HTTP boot driver private data.

Return values:

EFI_SUCCESS The D.O.R.A process successfully finished.
Others Failed to finish the D.O.R.A process.

References HTTP_BOOT_DHCP4_OPTION_MAX_NUM, HTTP_BOOT_DHCP4_OPTION_MAX_SIZE, HTTP_BOOT_DHCP_RETRIES, HttpBootBuildDhcp4Options(), HttpBootDhcp4CallBack(), HttpBootRegisterIp4Gateway(), HttpBootSetIpPolicy(), HttpBootShowIp4Addr(), and mHttpDhcpTimeout.

Referenced by HttpBootDhcp().

EFI_DHCP4_PACKET_OPTION* HttpBootParseDhcp4Options (IN UINT8 * *Buffer*,
IN UINT32 *Length*,
IN UINT8 *OptTag*
)

Parse a certain dhcp4 option by OptTag in Buffer, and return with start pointer.

Parameters:

[in] *Buffer* Pointer to the option buffer.
[in] *Length* Length of the option buffer.
[in] *OptTag* Tag of the required option.

Return values:

NULL Failed to find the required option.
Others The position of the required option.

References HTTP_BOOT_DHCP4_TAG_EOP, and HTTP_BOOT_DHCP4_TAG_PAD.

Referenced by HttpBootDhcp4CallBack(), and HttpBootParseDhcp4Packet().

EFI_STATUS HttpBootParseDhcp4Packet (IN HTTP_BOOT_DHCP4_PACKET_CACHE * *Cache4*)

Parse the cached DHCPv4 packet, including all the options.

Parameters:

[in] *Cache4* Pointer to cached DHCPv4 packet.

Return values:

EFI_SUCCESS	Parsed the DHCPv4 packet successfully.
EFI_DEVICE_ERROR	Failed to parse an invalid packet.

References DEFAULT_CLASS_ID_DATA, GET_OPTION_BUFFER_LEN, HTTP_BOOT_DHCP4_OVERLOAD_FILE, HTTP_BOOT_DHCP4_OVERLOAD_SERVER_NAME, HTTP_BOOT_DHCP4_TAG_INDEX_BOOTFILE, HTTP_BOOT_DHCP4_TAG_INDEX_CLASS_ID, HTTP_BOOT_DHCP4_TAG_INDEX_DNS_SERVER, HTTP_BOOT_DHCP4_TAG_INDEX_MAX, HTTP_BOOT_DHCP4_TAG_INDEX_OVERLOAD, HttpBootParseDhcp4Options(), HttpOfferTypeDhcpDns, HttpOfferTypeDhcpIpUri, HttpOfferTypeDhcpNameUri, HttpOfferTypeDhcpNameUriDns, HttpOfferTypeDhcpOnly, HttpOfferTypeProxyIpUri, HttpOfferTypeProxyNameUri, and mInterestedDhcp4Tags.

Referenced by HttpBootCacheDhcp4Offer().

```
EFI_STATUS HttpBootRegisterIp4Dns ( IN HTTP_BOOT_PRIVATE_DATA *   Private,
                                    IN UINTN                      DataLength,
                                    IN VOID *                     DnsServerData
                                  )
```

This function will register the default DNS addresses to the network device.

Parameters:

[in]	*Private*	The pointer to HTTP_BOOT_PRIVATE_DATA.
[in]	*DataLength*	Size of the buffer pointed to by DnsServerData in bytes.
[in]	*DnsServerData*	Point a list of DNS server address in an array of EFI_IPv4_ADDRESS instances.

Return values:

EFI_SUCCESS	The DNS configuration has been configured successfully.
Others	Failed to configure the address.

Referenced by HttpBootExtractUriInfo().

```
EFI_STATUS HttpBootRegisterIp4Gateway ( IN HTTP_BOOT_PRIVATE_DATA *   Private   )
```

This function will register the IPv4 gateway address to the network device.

Parameters:

[in]	*Private*	The pointer to HTTP_BOOT_PRIVATE_DATA.

Return values:

EFI_SUCCESS	The new IP configuration has been configured successfully.
Others	Failed to configure the address.

Referenced by HttpBootDhcp4Dora().

```
VOID HttpBootSelectDhcp4Offer ( IN HTTP_BOOT_PRIVATE_DATA *   Private   )
```

Select an DHCPv4 offer, and record SelectIndex and SelectProxyType.

Parameters:

[in]	*Private*	Pointer to HTTP boot driver private data.

References HttpOfferTypeDhcpDns, HttpOfferTypeDhcpIpUri, HttpOfferTypeDhcpNameUri, HttpOfferTypeDhcpNameUriDns, HttpOfferTypeDhcpOnly, HttpOfferTypeMax, HttpOfferTypeProxyIpUri, and HttpOfferTypeProxyNameUri.

Referenced by HttpBootDhcp4CallBack().

EFI_STATUS HttpBootSetIpPolicy (IN HTTP_BOOT_PRIVATE_DATA * *Private*)

This function will switch the IP4 configuration policy to Static.

Parameters:

`[in]` *Private* Pointer to HTTP boot driver private data.

Return values:

EFI_SUCCESS The policy is already configured to static.

Others Other error as indicated..

References Policy.

Referenced by HttpBootDhcp4Dora().

Variable Documentation

UINT32 mHttpDhcpTimeout[4] = {4, 8, 16, 32}

Referenced by HttpBootDhcp4Dora().

UINT8 mInterestedDhcp4Tags[HTTP_BOOT_DHCP4_TAG_INDEX_MAX]

Initial value:

```
{
  HTTP_BOOT_DHCP4_TAG_BOOTFILE_LEN,
  HTTP_BOOT_DHCP4_TAG_OVERLOAD,
  HTTP_BOOT_DHCP4_TAG_MSG_TYPE,
  HTTP_BOOT_DHCP4_TAG_SERVER_ID,
  HTTP_BOOT_DHCP4_TAG_CLASS_ID,
  HTTP_BOOT_DHCP4_TAG_BOOTFILE,
  HTTP_BOOT_DHCP4_TAG_DNS_SERVER
}
```

Referenced by HttpBootParseDhcp4Packet(), and PxeBcParseDhcp4Packet().

NetworkPkg/HttpBootDxe/HttpBootDhcp4.h File Reference

Data Structures

struct HTTP_BOOT_DHCP4_OPT
struct HTTP_BOOT_DHCP4_OPT
struct HTTP_BOOT_DHCP4_OPT
struct HTTP_BOOT_DHCP4_OPT
struct HTTP_BOOT_DHCP4_OPT
struct HTTP_BOOT_DHCP4_OPT
struct HTTP_BOOT_DHCP4_OPT
struct HTTP_BOOT_OPTION_BO
union HTTP_BOOT_DHCP4_OPT
struct HTTP_BOOT_VENDOR_OI
union HTTP_BOOT_DHCP4_PAC
struct HTTP_BOOT_URI_CONTE
struct HTTP_BOOT_DHCP4_PAC

Defines

"memItemRight" valign="bottom">HTTP_BOOT_DHCP4_OPTION_MAX_NUM 16

#define HTTP_BOOT_DHCP4_OPT
#define HTTP_BOOT_DHCP4_PAC
#define HTTP_BOOT_DHCP4_OPC
#define HTTP_BOOT_DHCP4_OPC
#define HTTP_BOOT_DHCP4_MSC
#define HTTP_BOOT_DHCP4_MAC
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG
#define HTTP_BOOT_DHCP4_TAG

```
#define HTTP_BOOT_DHCP4_TAG_OVER
#define HTTP_BOOT_DHCP4_TAG_MSG_
#define HTTP_BOOT_DHCP4_TAG_SERV
#define HTTP_BOOT_DHCP4_TAG_PARA
#define HTTP_BOOT_DHCP4_TAG_MAXM
#define HTTP_BOOT_DHCP4_TAG_T1   58
#define HTTP_BOOT_DHCP4_TAG_T2   59
#define HTTP_BOOT_DHCP4_TAG_CLAS
#define HTTP_BOOT_DHCP4_TAG_CLIEN
#define HTTP_BOOT_DHCP4_TAG_TFTP
#define HTTP_BOOT_DHCP4_TAG_BOOT
#define HTTP_BOOT_DHCP4_TAG_ARCH
#define HTTP_BOOT_DHCP4_TAG_UNDI
#define HTTP_BOOT_DHCP4_TAG_UUID
#define HTTP_BOOT_DHCP4_OVERLOAD
#define HTTP_BOOT_DHCP4_OVERLOAD
#define EFI_HTTP_BOOT_CLIENT_SYSTE
#define HTTP_BOOT_DHCP_RETRIES   4
#define HTTP_BOOT_OFFER_MAX_NUM
#define HTTP_BOOT_DHCP4_TAG_INDEX
#define HTTP_BOOT_DHCP4_TAG_INDEX
#define HTTP_BOOT_DHCP4_TAG_INDEX
#define HTTP_BOOT_DHCP4_TAG_INDEX
#define HTTP_BOOT_DHCP4_TAG_INDEX
#define HTTP_BOOT_DHCP4_TAG_INDEX
#define HTTP_BOOT_DHCP4_TAG_INDEX
#define HTTP_BOOT_DHCP4_TAG_INDEX
#define GET_NEXT_DHCP_OPTION(Opt)
#define GET_OPTION_BUFFER_LEN(Pkt)
        (EFI_DHCP4_HEADER) - 4)
#define DEFAULT_CLASS_ID_DATA   "HT
#define DEFAULT_UNDI_TYPE   1
#define DEFAULT_UNDI_MAJOR   3
#define DEFAULT_UNDI_MINOR   0
```

Enumerations

```
enum HTTP_BOOT_OFFER_TYPE {
       HttpOfferTypeDhcpIpUri, HttpOffer
      HttpOfferTypeDhcpDns, HttpOfferTy
       HttpOfferTypeProxyNameUri, Http
      HttpOfferTypeDhcpNameUri, HttpOf
      }
```

Functions

```
EFI_STATUS HttpBootDhcp4Dora (IN HTTP_BOO
EFI_STATUS HttpBootRegisterIp4Dns (IN HTTP_
      UINTN DataLength, IN VOID *DnsS
```

Detailed Description

Functions declaration related with DHCPv4 for HTTP boot driver.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by HttpBootBuildDhcp4Options(), HttpBootParseDhcp4Packet(), PxeBcBuildDhcp4Options(), PxeBcBuildDhcp6Options(), PxeBcParseDhcp4Packet(), and PxeBcParseDhcp6Packet().

#define DEFAULT_UNDI_MAJOR 3

Referenced by HttpBootBuildDhcp4Options(), PxeBcBuildDhcp4Options(), and PxeBcBuildDhcp6Options().

#define DEFAULT_UNDI_MINOR 0

Referenced by HttpBootBuildDhcp4Options(), PxeBcBuildDhcp4Options(), and PxeBcBuildDhcp6Options().

#define DEFAULT_UNDI_TYPE 1

Referenced by HttpBootBuildDhcp4Options(), PxeBcBuildDhcp4Options(), and PxeBcBuildDhcp6Options().

#define EFI_HTTP_BOOT_CLIENT_SYSTEM_ARCHITECTURE 0x000F

HTTP Tag definition that identifies the processor and programming environment of the client system. These identifiers are defined by IETF: http://www.ietf.org/assignments/dhcpv6-parameters/dhcpv6-parameters.xml

Referenced by HttpBootBuildDhcp4Options().

#define GET_NEXT_DHCP_OPTION (Opt)

Value:

```
(EFI_DHCP4_PACKET_OPTION *) ((UINT8 *) (Opt) + \
   sizeof (EFI_DHCP4_PACKET_OPTION) + (Opt)->Length - 1)
```

Referenced by HttpBootBuildDhcp4Options(), and PxeBcBuildDhcp4Options().

#define GET_OPTION_BUFFER_LEN (Pkt) ((Pkt)->Length - sizeof (EFI_DHCP4_HEADER) - 4)

Referenced by HttpBootDhcp4CallBack(), HttpBootParseDhcp4Packet(), PxeBcDhcp4CallBack(), and PxeBcParseDhcp4Packet().

#define HTTP_BOOT_DHCP4_MAGIC 0x63538263

#define HTTP_BOOT_DHCP4_MSG_TYPE_REQUEST 3

#define HTTP_BOOT_DHCP4_OPCODE_REPLY 2

#define HTTP_BOOT_DHCP4_OPCODE_REQUEST 1

#define HTTP_BOOT_DHCP4_OPTION_MAX_NUM 16

Referenced by HttpBootDhcp4Dora().

#define HTTP_BOOT_DHCP4_OPTION_MAX_SIZE 312

Referenced by HttpBootDhcp4Dora().

#define HTTP_BOOT_DHCP4_OVERLOAD_FILE 1

Referenced by HttpBootParseDhcp4Packet().

#define HTTP_BOOT_DHCP4_OVERLOAD_SERVER_NAME 2

Referenced by HttpBootParseDhcp4Packet().

#define HTTP_BOOT_DHCP4_PACKET_MAX_SIZE 1472

Referenced by HttpBootDhcp4CallBack(), and HttpBootStart().

#define HTTP_BOOT_DHCP4_TAG_ARCH 93

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_BOOTFILE 67

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_BOOTFILE_LEN 13

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_BROADCAST 28

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_CLASS_ID 60

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_CLIENT_ID 61

#define HTTP_BOOT_DHCP4_TAG_DNS_SERVER 6

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_DOMAINNAME 15

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_DUMP 14

#define HTTP_BOOT_DHCP4_TAG_EMTU 22

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_EOP 255

Referenced by HttpBootParseDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_EXTEND_PATH 18

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_HOSTNAME 12

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_INDEX_BOOTFILE 5

Referenced by HttpBootExtractUriInfo(), and HttpBootParseDhcp4Packet().

#define HTTP_BOOT_DHCP4_TAG_INDEX_BOOTFILE_LEN 0

#define HTTP_BOOT_DHCP4_TAG_INDEX_CLASS_ID 4

Referenced by HttpBootParseDhcp4Packet().

#define HTTP_BOOT_DHCP4_TAG_INDEX_DNS_SERVER 6

Referenced by HttpBootExtractUriInfo(), and HttpBootParseDhcp4Packet().

#define HTTP_BOOT_DHCP4_TAG_INDEX_MAX 7

Referenced by HttpBootParseDhcp4Packet().

#define HTTP_BOOT_DHCP4_TAG_INDEX_MSG_TYPE 2

#define HTTP_BOOT_DHCP4_TAG_INDEX_OVERLOAD 1

Referenced by HttpBootParseDhcp4Packet().

#define HTTP_BOOT_DHCP4_TAG_INDEX_SERVER_ID 3

#define HTTP_BOOT_DHCP4_TAG_LEASE 51

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_MAXMSG 57

Referenced by HttpBootDhcp4CallBack().

#define HTTP_BOOT_DHCP4_TAG_MSG_TYPE 53

#define HTTP_BOOT_DHCP4_TAG_NAME_SERVER 5

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_NETMASK 1

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_NIS_DOMAIN 40

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_NIS_SERVER 41

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_NTP_SERVER 42

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_OVERLOAD 52

#define HTTP_BOOT_DHCP4_TAG_PAD 0

Referenced by HttpBootParseDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_PARA_LIST 55

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_REQUEST_IP 50

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_ROOTPATH 17

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_ROUTER 3

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_SERVER_ID 54

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_T1 58

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_T2 59

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_TFTP 66

#define HTTP_BOOT_DHCP4_TAG_TIME_OFFSET 2

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_TIME_SERVER 4

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_TTL 23

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_UNDI 94

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_UUID 97

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP4_TAG_VENDOR 43

Referenced by HttpBootBuildDhcp4Options().

#define HTTP_BOOT_DHCP_RETRIES 4

Referenced by HttpBootDhcp4Dora().

#define HTTP_BOOT_OFFER_MAX_NUM 16

Referenced by HttpBootCacheDhcp4Offer(), HttpBootDhcp4CallBack(), HttpBootExtractUriInfo(), HttpBootStart(), and HttpBootStop().

Enumeration Type Documentation

enum HTTP_BOOT_OFFER_TYPE

DHCP offer types among HTTP boot. Dhcp4 and Dhcp6 share this definition, and corresponding relatioinship is as follows: Dhcp4Discover <> Dhcp6Solicit Dhcp4Offer <> Dhcp6Advertise Dhcp4Request <> Dhcp6Request Dhcp4Ack <> DHcp6Reply

Enumerator:

HttpOfferTypeDhcpIpUri
HttpOfferTypeDhcpNameUriDns
HttpOfferTypeDhcpDns
HttpOfferTypeDhcpOnly
HttpOfferTypeProxyNameUri
HttpOfferTypeProxyIpUri
HttpOfferTypeDhcpNameUri
HttpOfferTypeMax

Function Documentation

EFI_STATUS HttpBootDhcp4Dora (IN HTTP_BOOT_PRIVATE_DATA * *Private*)

Start the D.O.R.A DHCPv4 process to acquire the IPv4 address and other Http boot information.

Parameters:

`[in]` *Private* Pointer to HTTP_BOOT private data.

Return values:

EFI_SUCCESS The D.O.R.A process successfully finished.
Others Failed to finish the D.O.R.A process.

Start the D.O.R.A DHCPv4 process to acquire the IPv4 address and other Http boot information.

Parameters:

`[in]` *Private* Pointer to HTTP boot driver private data.

Return values:

EFI_SUCCESS The D.O.R.A process successfully finished.
Others Failed to finish the D.O.R.A process.

References HTTP_BOOT_DHCP4_OPTION_MAX_NUM, HTTP_BOOT_DHCP4_OPTION_MAX_SIZE, HTTP_BOOT_DHCP_RETRIES, HttpBootBuildDhcp4Options(), HttpBootDhcp4CallBack(), HttpBootRegisterIp4Gateway(), HttpBootSetIpPolicy(), HttpBootShowIp4Addr(), and mHttpDhcpTimeout.

Referenced by HttpBootDhcp().

```
EFI_STATUS HttpBootRegisterIp4Dns ( IN HTTP_BOOT_PRIVATE_DATA *   Private,
                                    IN UINTN                      DataLength,
                                    IN VOID *                     DnsServerData
                                  )
```

This function will register the default DNS addresses to the network device.

Parameters:

[in]	*Private*	The pointer to HTTP_BOOT_PRIVATE_DATA.
[in]	*DataLength*	Size of the buffer pointed to by DnsServerData in bytes.
[in]	*DnsServerData*	Point a list of DNS server address in an array of EFI_IPv4_ADDRESS instances.

Return values:

EFI_SUCCESS	The DNS configuration has been configured successfully.
Others	Failed to configure the address.

Referenced by HttpBootExtractUriInfo().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/HttpBootDxe/HttpBootDxe.c File Reference

Functions

VOID	HttpBootDestroyIp4Children (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN HTTP_BOOT_PRIVATE_DATA *Private)
EFI_STATUS EFIAPI	HttpBootIp4DxeDriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI	HttpBootIp4DxeDriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI	HttpBootIp4DxeDriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)
EFI_STATUS EFIAPI	HttpBootDxeDriverEntryPoint (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)

Variables

EFI_DRIVER_BINDING_PROTOCOL gHttpBootIp4DxeDriverBinding

Detailed Description

Driver Binding functions implementation for UEFI HTTP boot.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
VOID HttpBootDestroyIp4Children ( IN EFI_DRIVER_BINDING_PROTOCOL *   This,
                                  IN HTTP_BOOT_PRIVATE_DATA *        Private
                                )
```

Destroy the HTTP child based on IPv4 stack.

Parameters:

[in] *This* Pointer to the EFI_DRIVER_BINDING_PROTOCOL.
[in] *Private* Pointer to HTTP_BOOT_PRIVATE_DATA.

References HttpIoDestroyIo().

Referenced by HttpBootIp4DxeDriverBindingStart(), and HttpBootIp4DxeDriverBindingStop().

```
EFI_STATUS EFIAPI HttpBootDxeDriverEntryPoint ( IN EFI_HANDLE          ImageHandle,
                                                IN EFI_SYSTEM_TABLE *  SystemTable
                                              )
```

This is the declaration of an EFI image entry point. This entry point is the same for UEFI Applications, UEFI OS Loaders, and UEFI Drivers including both device drivers and bus drivers.

Parameters:
[in] *ImageHandle* The firmware allocated handle for the UEFI image.
[in] *SystemTable* A pointer to the EFI System Table.

Return values:
EFI_SUCCESS The operation completed successfully.
Others An unexpected error occurred.

References gHttpBootDxeComponentName, gHttpBootDxeComponentName2, and gHttpBootIp4DxeDriverBinding.

```
EFI_STATUS EFIAPI
HttpBootIp4DxeDriverBindingStart ( IN EFI_DRIVER_BINDING_PROTOCOL *  This,
                                   IN EFI_HANDLE                     ControllerHandle,
                                   IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
                                 )
```

Starts a device controller or a bus controller.

The Start() function is designed to be invoked from the EFI boot service ConnectController(). As a result, much of the error checking on the parameters to Start() has been moved into this common boot service. It is legal to call Start() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE. 2. If RemainingDevicePath is not NULL, then it must be a pointer to a naturally aligned EFI_DEVICE_PATH_PROTOCOL. 3. Prior to calling Start(), the Supported() function for the driver specified by This must have been called with the same calling parameters, and Supported() must have returned EFI_SUCCESS.

Parameters:
[in] *This* A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in] *ControllerHandle* The handle of the controller to start. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in] *RemainingDevicePath* A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For a bus driver, if this parameter is NULL, then handles for all the children of Controller are created by this driver. If this parameter is not NULL and

the first Device Path Node is not the End of Device Path Node, then only the handle for the child device specified by the first Device Path Node of RemainingDevicePath is created by this driver. If the first Device Path Node of RemainingDevicePath is the End of Device Path Node, no child handle is created by this driver.

Return values:

EFI_SUCCESS	The device was started.
EFI_DEVICE_ERROR	The device could not be started due to a device error.Currently not implemented.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
Others	The driver failded to start the device.

References _HTTP_BOOT_PRIVATE_DATA::CacheList, _HTTP_BOOT_PRIVATE_DATA::ChildHandle, _HTTP_BOOT_PRIVATE_DATA::Controller, _HTTP_BOOT_PRIVATE_DATA::DevicePath, _HTTP_BOOT_PRIVATE_DATA::Dhcp4, _HTTP_BOOT_PRIVATE_DATA::Dhcp4Child, gHttpBootDxeLoadFile, HTTP_BOOT_PRIVATE_DATA_SIGNATURE, HttpBootDestroyIp4Children(), _HTTP_BOOT_PRIVATE_DATA::Id, _HTTP_BOOT_PRIVATE_DATA::Image, _HTTP_BOOT_PRIVATE_DATA::Ip4Config2, _HTTP_BOOT_PRIVATE_DATA::LoadFile, _HTTP_BOOT_PRIVATE_DATA::Nii, _HTTP_BOOT_PRIVATE_DATA::ParentDevicePath, _HTTP_BOOT_PRIVATE_DATA::Signature, and _HTTP_BOOT_PRIVATE_DATA::UsingIpv6.

```
EFI_STATUS EFIAPI
HttpBootIp4DxeDriverBindingStop
  ( IN EFI_DRIVER_BINDING_PROTOCOL *   This,
    IN EFI_HANDLE                      ControllerHandle,
    IN UINTN                           NumberOfChildren,
    IN EFI_HANDLE *                    ChildHandleBuffer
  )
```

Stops a device controller or a bus controller.

The Stop() function is designed to be invoked from the EFI boot service DisconnectController(). As a result, much of the error checking on the parameters to Stop() has been moved into this common boot service. It is legal to call Stop() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE that was used on a previous call to this same driver's Start() function. 2. The first NumberOfChildren handles of ChildHandleBuffer must all be a valid EFI_HANDLE. In addition, all of these handles must have been created in this driver's Start() function, and the Start() function must have called OpenProtocol() on ControllerHandle with an Attribute of EFI_OPEN_PROTOCOL_BY_CHILD_CONTROLLER.

Parameters:

`[in]`	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
`[in]`	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
`[in]`	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
`[in]`	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References _HTTP_BOOT_PRIVATE_DATA::Controller, HTTP_BOOT_PRIVATE_DATA_FROM_ID, HTTP_BOOT_PRIVATE_DATA_FROM_LOADFILE, HttpBootDestroyIp4Children(), HttpBootFreeCacheList(), HttpBootGetNicByIp4Children(), HttpBootStop(), and _HTTP_BOOT_PRIVATE_DATA::Id.

```
EFI_STATUS EFIAPI
HttpBootIp4DxeDriverBindingSupported (
  IN EFI_DRIVER_BINDING_PROTOCOL *        This,
  IN EFI_HANDLE                           ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL *           RemainingDevicePath
)
```

Tests to see if this driver supports a given controller. If a child device is provided, it further tests to see if this driver supports creating a handle for the specified child device.

This function checks to see if the driver specified by This supports the device specified by ControllerHandle. Drivers will typically use the device path attached to ControllerHandle and/or the services from the bus I/O abstraction attached to ControllerHandle to determine if the driver supports ControllerHandle. This function may be called many times during platform initialization. In order to reduce boot times, the tests performed by this function must be very small, and take as little time as possible to execute. This function must not change the state of any hardware devices, and this function must be aware that the device specified by ControllerHandle may already be managed by the same driver or a different driver. This function must match its calls to AllocatePages() with FreePages(), AllocatePool() with FreePool(), and OpenProtocol() with CloseProtocol(). Because ControllerHandle may have been previously started by the same driver, if a protocol is already in the opened state, then it must not be closed with CloseProtocol(). This is required to guarantee the state of ControllerHandle is not modified by this function.

Parameters:

`[in]`	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
`[in]`	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
`[in]`	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by the driver specified by This.
EFI_ACCESS_DENIED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by a different driver or an application that requires exclusive access. Currently not implemented.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

Variable Documentation

EFI_DRIVER_BINDING_PROTOCOL gHttpBootIp4DxeDriverBinding

Initial value:

```
{
  HttpBootIp4DxeDriverBindingSupported,
  HttpBootIp4DxeDriverBindingStart,
  HttpBootIp4DxeDriverBindingStop,
  HTTP_BOOT_DXE_VERSION,
  NULL,
  NULL
}
```

Driver Binding Protocol instance

Referenced by HttpBootDxeDriverEntryPoint().

NetworkPkg/HttpBootDxe/HttpBootDxe.h File Reference

Data Structures

union HTTP_BOOT_DHCP_PACKET_CACHE
struct _HTTP_BOOT_PRIVATE_DATA

Defines

"memItemRight" valign="bottom">HTTP_BOOT_DXE_VERSION 0xa
#define HTTP_BOOT_PRIVATE_DATA_SIGNAT
('H', 'B', 'P', 'D')
#define HTTP_BOOT_PRIVATE_DATA_FROM_
(a, HTTP_BOOT_PRIVATE_DATA, Load
HTTP_BOOT_PRIVATE_DATA_SIGNAT
#define HTTP_BOOT_PRIVATE_DATA_FROM_
(a, HTTP_BOOT_PRIVATE_DATA, Id,
HTTP_BOOT_PRIVATE_DATA_SIGNAT

Typedefs

typedef struct
_HTTP_BOOT_PRIVATE_DATA HTTP_BOOT_PRIVATE_DATA

Functions

EFI_STATUS EFIAPI HttpBootIp4DxeDriverBindingSupported (I
EFI_DRIVER_BINDING_PROTOCOL *T
EFI_HANDLE ControllerHandle, IN
EFI_DEVICE_PATH_PROTOCOL
*RemainingDevicePath)
EFI_STATUS EFIAPI HttpBootIp4DxeDriverBindingStart (IN
EFI_DRIVER_BINDING_PROTOCOL *T
EFI_HANDLE ControllerHandle, IN
EFI_DEVICE_PATH_PROTOCOL
*RemainingDevicePath)
EFI_STATUS EFIAPI HttpBootIp4DxeDriverBindingStop (IN
EFI_DRIVER_BINDING_PROTOCOL *T
EFI_HANDLE ControllerHandle, IN UINT
NumberOfChildren, IN EFI_HANDLE
*ChildHandleBuffer)

Variables

EFI_DRIVER_BINDING_PROTOCOL gHttpBootDxeDriverBinding
EFI_COMPONENT_NAME2_PROTOCOL gHttpBootDxeComponentName2
EFI_COMPONENT_NAME_PROTOCOL gHttpBootDxeComponentName
EFI_LOAD_FILE_PROTOCOL gHttpBootDxeLoadFile

Detailed Description

UEFI HTTP boot driver's private data structure and interfaces declaration.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

#define HTTP_BOOT_PRIVATE_DATA_FROM_ID (a) CR (a, HTTP_BOOT_PRIVATE_DATA, Id, HTTP_BOOT_PRIVATE_DATA_SIGNATURE)

Referenced by HttpBootIp4DxeDriverBindingStop().

#define HTTP_BOOT_PRIVATE_DATA_FROM_LOADFILE (a) CR (a, HTTP_BOOT_PRIVATE_DATA, LoadFile, HTTP_BOOT_PRIVATE_DATA_SIGNATURE)

Referenced by HttpBootDxeLoadFile(), and HttpBootIp4DxeDriverBindingStop().

#define HTTP_BOOT_PRIVATE_DATA_SIGNATURE SIGNATURE_32 ('H', 'B', 'P', 'D')

Referenced by HttpBootIp4DxeDriverBindingStart().

Typedef Documentation

typedef struct _HTTP_BOOT_PRIVATE_DATA HTTP_BOOT_PRIVATE_DATA

Function Documentation

```
EFI_STATUS EFIAPI
HttpBootIp4DxeDriverBindingStart (
    IN EFI_DRIVER_BINDING_PROTOCOL *   This,
    IN EFI_HANDLE                      ControllerHandle,
    IN EFI_DEVICE_PATH_PROTOCOL *      RemainingDevicePath
)
```

Starts a device controller or a bus controller.

The Start() function is designed to be invoked from the EFI boot service ConnectController(). As a result, much of the error checking on the parameters to Start() has been moved into this common boot service. It is legal to call Start() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE. 2. If RemainingDevicePath is not NULL, then it must be a pointer to a naturally aligned EFI_DEVICE_PATH_PROTOCOL. 3. Prior to calling Start(), the Supported() function for the driver specified by This must have been called with the same calling parameters, and Supported() must have returned EFI_SUCCESS.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to start. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For a bus driver, if this parameter is NULL, then handles for all the children of Controller are created by this driver. If this parameter is not NULL and the first Device Path Node is not the End of Device Path Node, then only the handle for the child device specified by the first Device Path Node of RemainingDevicePath is created by this driver. If the first Device Path Node of RemainingDevicePath is the End of Device Path Node, no child handle is created by this driver.

Return values:

EFI_SUCCESS	The device was started.
EFI_DEVICE_ERROR	The device could not be started due to a device error.Currently not implemented.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
Others	The driver failded to start the device.

References _HTTP_BOOT_PRIVATE_DATA::CacheList, _HTTP_BOOT_PRIVATE_DATA::ChildHandle, _HTTP_BOOT_PRIVATE_DATA::Controller, _HTTP_BOOT_PRIVATE_DATA::DevicePath, _HTTP_BOOT_PRIVATE_DATA::Dhcp4, _HTTP_BOOT_PRIVATE_DATA::Dhcp4Child, gHttpBootDxeLoadFile, HTTP_BOOT_PRIVATE_DATA_SIGNATURE, HttpBootDestroyIp4Children(), _HTTP_BOOT_PRIVATE_DATA::Id, _HTTP_BOOT_PRIVATE_DATA::Image, _HTTP_BOOT_PRIVATE_DATA::Ip4Config2, _HTTP_BOOT_PRIVATE_DATA::LoadFile, _HTTP_BOOT_PRIVATE_DATA::Nii, _HTTP_BOOT_PRIVATE_DATA::ParentDevicePath, _HTTP_BOOT_PRIVATE_DATA::Signature, and _HTTP_BOOT_PRIVATE_DATA::UsingIpv6.

```
EFI_STATUS EFIAPI
HttpBootIp4DxeDriverBindingStop
  ( IN EFI_DRIVER_BINDING_PROTOCOL *   This,
    IN EFI_HANDLE                      ControllerHandle,
    IN UINTN                           NumberOfChildren,
    IN EFI_HANDLE *                    ChildHandleBuffer
  )
```

Stops a device controller or a bus controller.

The Stop() function is designed to be invoked from the EFI boot service DisconnectController(). As a result, much of the error checking on the parameters to Stop() has been moved into this common boot service. It is

legal to call Stop() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE that was used on a previous call to this same driver's Start() function. 2. The first NumberOfChildren handles of ChildHandleBuffer must all be a valid EFI_HANDLE. In addition, all of these handles must have been created in this driver's Start() function, and the Start() function must have called OpenProtocol() on ControllerHandle with an Attribute of EFI_OPEN_PROTOCOL_BY_CHILD_CONTROLLER.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
[in]	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
[in]	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References _HTTP_BOOT_PRIVATE_DATA::Controller, HTTP_BOOT_PRIVATE_DATA_FROM_ID, HTTP_BOOT_PRIVATE_DATA_FROM_LOADFILE, HttpBootDestroyIp4Children(), HttpBootFreeCacheList(), HttpBootGetNicByIp4Children(), HttpBootStop(), and _HTTP_BOOT_PRIVATE_DATA::Id.

```
EFI_STATUS EFIAPI
HttpBootIp4DxeDriverBindingSupported ( IN EFI_DRIVER_BINDING_PROTOCOL *  This,
                                       IN EFI_HANDLE                     ControllerHandle,
                                       IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
                                       )
```

Tests to see if this driver supports a given controller. If a child device is provided, it further tests to see if this driver supports creating a handle for the specified child device.

This function checks to see if the driver specified by This supports the device specified by ControllerHandle. Drivers will typically use the device path attached to ControllerHandle and/or the services from the bus I/O abstraction attached to ControllerHandle to determine if the driver supports ControllerHandle. This function may be called many times during platform initialization. In order to reduce boot times, the tests performed by this function must be very small, and take as little time as possible to execute. This function must not change the state of any hardware devices, and this function must be aware that the device specified by ControllerHandle may already be managed by the same driver or a different driver. This function must match its calls to AllocatePages() with FreePages(), AllocatePool() with FreePool(), and OpenProtocol() with CloseProtocol(). Because ControllerHandle may have been previously started by the same driver, if a protocol is already in the opened state, then it must not be closed with CloseProtocol(). This is required to guarantee the state of ControllerHandle is not modified by this function.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus

drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by the driver specified by This.
EFI_ACCESS_DENIED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by a different driver or an application that requires exclusive access. Currently not implemented.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

Variable Documentation

EFI_COMPONENT_NAME_PROTOCOL gHttpBootDxeComponentName
Component Name Protocol instance

Referenced by HttpBootDxeDriverEntryPoint().

EFI_COMPONENT_NAME2_PROTOCOL gHttpBootDxeComponentName2
Component Name 2 Protocol instance

Referenced by HttpBootDxeComponentNameGetControllerName(), HttpBootDxeComponentNameGetDriverName(), and HttpBootDxeDriverEntryPoint().

EFI_DRIVER_BINDING_PROTOCOL gHttpBootDxeDriverBinding

EFI_LOAD_FILE_PROTOCOL gHttpBootDxeLoadFile
Load File Protocol instance

Referenced by HttpBootIp4DxeDriverBindingStart().

NetworkPkg/HttpBootDxe/HttpBootImpl.c File Reference

Functions

EFI_STATUS HttpBootStart (IN HTTP_BOOT_PRIVATE_DATA *Private)
EFI_STATUS HttpBootDhcp (IN HTTP_BOOT_PRIVATE_DATA *Private)
EFI_STATUS HttpBootLoadFile (IN HTTP_BOOT_PRIVATE_DATA *Private, IN OUT UINTN *BufferSize, IN VOID *Buffer)
EFI_STATUS HttpBootStop (IN HTTP_BOOT_PRIVATE_DATA *Private)
EFI_STATUS EFIAPI HttpBootDxeLoadFile (IN EFI_LOAD_FILE_PROTOCOL *This, IN EFI_DEVICE_PATH_PROTOCOL *FilePath, IN BOOLEAN BootPolicy, IN OUT UINTN *BufferSize, IN VOID *Buffer)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED
EFI_LOAD_FILE_PROTOCOL gHttpBootDxeLoadFile

Detailed Description

The implementation of EFI_LOAD_FILE_PROTOCOL for UEFI HTTP boot.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS HttpBootDhcp (IN HTTP_BOOT_PRIVATE_DATA * *Private*)

Attempt to complete a DHCPv4 D.O.R.A sequence to retrieve the boot resource information.

Parameters:

[in]	*Private*	The pointer to the driver's private data.

Return values:

EFI_SUCCESS	Boot info was successfully retrieved.
EFI_INVALID_PARAMETER	Private is NULL.
EFI_NOT_STARTED	The driver is in stopped state.
EFI_DEVICE_ERROR	An unexpected network error occurred.
Others	Other errors as indicated.

References HttpBootDhcp4Dora().

Referenced by HttpBootDiscoverBootInfo().

```
EFI_STATUS EFIAPI HttpBootDxeLoadFile ( IN EFI_LOAD_FILE_PROTOCOL *        This,
                                        IN EFI_DEVICE_PATH_PROTOCOL *      FilePath,
                                        IN BOOLEAN                         BootPolicy,
                                        IN OUT UINTN *                     BufferSize,
                                        IN VOID *                          Buffer
                                      )
```

Causes the driver to load a specified file.

Parameters:

This	Protocol instance pointer.
FilePath	The device specific path of the file to load.
BootPolicy	If TRUE, indicates that the request originates from the boot manager is attempting to load FilePath as a boot selection. If FALSE, then FilePath must match as exact file to be loaded.
BufferSize	On input the size of Buffer in bytes. On output with a return code of EFI_SUCCESS, the amount of data transferred to Buffer. On output with a return code of EFI_BUFFER_TOO_SMALL, the size of Buffer required to retrieve the requested file.
Buffer	The memory buffer to transfer the file to. IF Buffer is NULL, then the size of the requested file is returned in BufferSize.

Return values:

EFI_SUCCESS	The file was loaded.
EFI_UNSUPPORTED	The device does not support the provided BootPolicy
EFI_INVALID_PARAMETER	FilePath is not a valid device path, or BufferSize is NULL.
EFI_NO_MEDIA	No medium was present to load the file.
EFI_DEVICE_ERROR	The file was not loaded due to a device error.
EFI_NO_RESPONSE	The remote system did not respond.
EFI_NOT_FOUND	The file was not found.
EFI_ABORTED	The file load process was manually cancelled.
EFI_BUFFER_TOO_SMALL	The BufferSize is too small to read the current directory entry. BufferSize has been updated with the size needed to complete the request.

References _HTTP_BOOT_PRIVATE_DATA::Controller, _HTTP_BOOT_PRIVATE_DATA::Dhcp4, HTTP_BOOT_PRIVATE_DATA_FROM_LOADFILE, HttpBootLoadFile(), HttpBootStart(), and HttpBootStop().

```
EFI_STATUS HttpBootLoadFile ( IN HTTP_BOOT_PRIVATE_DATA *   Private,
                              IN OUT UINTN *                BufferSize,
                              IN VOID *                     Buffer
                            )
```

Attempt to download the boot file through HTTP message exchange.

Parameters:

`[in]`	*Private*	The pointer to the driver's private data.

[in,out]	*BufferSize*	On input the size of Buffer in bytes. On output with a return code of EFI_SUCCESS, the amount of data transferred to Buffer. On output with a return code of EFI_BUFFER_TOO_SMALL, the size of Buffer required to retrieve the requested file.
[in]	*Buffer*	The memory buffer to transfer the file to. If Buffer is NULL, then the size of the requested file is returned in BufferSize.

Return values:

EFI_SUCCESS	Boot file was loaded successfully.
EFI_INVALID_PARAMETER	Private is NULL.
EFI_NOT_STARTED	The driver is in stopped state.
EFI_BUFFER_TOO_SMALL	The BufferSize is too small to read the boot file. BufferSize has been updated with the size needed to complete the request.
EFI_DEVICE_ERROR	An unexpected network error occurred.
Others	Other errors as indicated.

References HttpBootCreateHttpIo(), HttpBootDiscoverBootInfo(), and HttpBootGetBootFile().

Referenced by HttpBootDxeLoadFile().

EFI_STATUS HttpBootStart (IN HTTP_BOOT_PRIVATE_DATA * *Private*)

Enable the use of UEFI HTTP boot function.

Parameters:

[in]	*Private*	The pointer to the driver's private data.

Return values:

EFI_SUCCESS	HTTP boot was successfully enabled.
EFI_INVALID_PARAMETER	Private is NULL.
EFI_ALREADY_STARTED	The driver is already in started state.

References HTTP_BOOT_DHCP4_PACKET_MAX_SIZE, and HTTP_BOOT_OFFER_MAX_NUM.

Referenced by HttpBootDxeLoadFile().

EFI_STATUS HttpBootStop (IN HTTP_BOOT_PRIVATE_DATA * *Private*)

Disable the use of UEFI HTTP boot function.

Parameters:

[in]	*Private*	The pointer to the driver's private data.

Return values:

EFI_SUCCESS	HTTP boot was successfully disabled.
EFI_NOT_STARTED	The driver is already in stopped state.
EFI_INVALID_PARAMETER	Private is NULL.
Others	Unexpected error when stop the function.

References HTTP_BOOT_OFFER_MAX_NUM, HttpIoDestroyIo(), and HttpOfferTypeMax.

Referenced by HttpBootDxeLoadFile(), and HttpBootIp4DxeDriverBindingStop().

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED EFI_LOAD_FILE_PROTOCOL gHttpBootDxeLoadFile

Initial value:

```
{
 HttpBootDxeLoadFile
}
```

Load File Protocol instance

Referenced by HttpBootIp4DxeDriverBindingStart().

NetworkPkg/HttpBootDxe/HttpBootImpl.h File Reference

Functions

EFI_STATUS HttpBootDhcp (IN HTTP_BOOT_PRIVATE_DATA *Private)
EFI_STATUS HttpBootStop (IN HTTP_BOOT_PRIVATE_DATA *Private)

Detailed Description

The declaration of UEFI HTTP boot function.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS HttpBootDhcp (IN HTTP_BOOT_PRIVATE_DATA * *Private*)
Attempt to complete a DHCPv4 D.O.R.A sequence to retrieve the boot resource information.

Parameters:

[in]	*Private*	The pointer to the driver's private data.

Return values:

EFI_SUCCESS	Boot info was successfully retrieved.
EFI_INVALID_PARAMETER	Private is NULL.
EFI_NOT_STARTED	The driver is in stopped state.
EFI_DEVICE_ERROR	An unexpected network error occurred.
Others	Other errors as indicated.

References HttpBootDhcp4Dora().

Referenced by HttpBootDiscoverBootInfo().

EFI_STATUS HttpBootStop (IN HTTP_BOOT_PRIVATE_DATA * *Private*)
Disable the use of UEFI HTTP boot function.

Parameters:

[in]	*Private*	The pointer to the driver's private data.

Return values:

EFI_SUCCESS	HTTP boot was successfully disabled.
EFI_NOT_STARTED	The driver is already in stopped state.
EFI_INVALID_PARAMETER	Private is NULL.
Others	Unexpected error when stop the function.

References HTTP_BOOT_OFFER_MAX_NUM, HttpIoDestroyIo(), and HttpOfferTypeMax.

Referenced by HttpBootDxeLoadFile(), and HttpBootIp4DxeDriverBindingStop().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/HttpBootDxe/HttpBootSupport.c File Reference

Functions

EFI_HANDLE HttpBootGetNicByIp4Children (IN EFI_HANDLE ControllerHandle)
VOID HttpBootUintnToAscDecWithFormat (IN UINTN Number, IN UINT8 *Buffer, IN INTN Length)
VOID HttpBootShowIp4Addr (IN EFI_IPv4_ADDRESS *Ip)
HTTP_IO_HEADER * HttpBootCreateHeader (UINTN MaxHeaderCount)
VOID HttpBootFreeHeader (IN HTTP_IO_HEADER *HttpIoHeader)
EFI_HTTP_HEADER * HttpBootFindHeader (IN UINTN HeaderCount, IN EFI_HTTP_HEADER *Headers, IN CHAR8 *FieldName)
EFI_STATUS HttpBootSetHeader (IN HTTP_IO_HEADER *HttpIoHeader, IN CHAR8 *FieldName, IN CHAR8 *FieldValue)
VOID EFIAPI HttpIoCommonNotify (IN EFI_EVENT Event, IN VOID *Context)
EFI_STATUS HttpIoCreateIo (IN EFI_HANDLE Image, IN EFI_HANDLE Controller, IN UINT8 IpVersion, IN HTTP_IO_CONFIG_DATA *ConfigData, OUT HTTP_IO *HttpIo)
VOID HttpIoDestroyIo (IN HTTP_IO *HttpIo)
EFI_STATUS HttpIoSendRequest (IN HTTP_IO *HttpIo, IN EFI_HTTP_REQUEST_DATA *Request, IN UINTN HeaderCount, IN EFI_HTTP_HEADER *Headers, IN UINTN BodyLength, IN VOID *Body)
EFI_STATUS HttpIoRecvResponse (IN HTTP_IO *HttpIo, IN BOOLEAN RecvMsgHeader, OUT HTTP_IO_RESOPNSE_DATA *ResponseData)

Detailed Description

Support functions implementation for UEFI HTTP boot driver.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

HTTP_IO_HEADER* HttpBootCreateHeader (UINTN *MaxHeaderCount*)

Create a HTTP_IO_HEADER to hold the HTTP header items.

Parameters:

`[in]` *MaxHeaderCount* The maximun number of HTTP header in this holder.

Returns:

A pointer of the HTTP header holder or NULL if failed.

References HTTP_IO_HEADER::Headers, and HTTP_IO_HEADER::MaxHeaderCount.

```
EFI_HTTP_HEADER* HttpBootFindHeader ( IN UINTN                HeaderCount,
                                      IN EFI_HTTP_HEADER *    Headers,
                                      IN CHAR8 *              FieldName
                                    )
```

Find a specified header field according to the field name.

Parameters:

[in] *HeaderCount* Number of HTTP header structures in Headers list.
[in] *Headers* Array containing list of HTTP headers.
[in] *FieldName* Null terminated string which describes a field name.

Returns:

Pointer to the found header or NULL.

Referenced by HttpBootSetHeader().

```
VOID HttpBootFreeHeader ( IN HTTP_IO_HEADER *  HttpIoHeader  )
```

Destroy the HTTP_IO_HEADER and release the resouces.

Parameters:

[in] *HttpIoHeader* Point to the HTTP header holder to be destroyed.

Referenced by HttpBootGetBootFile().

```
EFI_HANDLE HttpBootGetNicByIp4Children ( IN EFI_HANDLE  ControllerHandle  )
```

Get the Nic handle using any child handle in the IPv4 stack.

Parameters:

[in] *ControllerHandle* Pointer to child handle over IPv4.

Returns:

NicHandle The pointer to the Nic handle.

NULL Can't find the Nic handle.

Referenced by HttpBootDxeComponentNameGetControllerName(), and HttpBootIp4DxeDriverBindingStop().

```
EFI_STATUS HttpBootSetHeader ( IN HTTP_IO_HEADER *  HttpIoHeader,
                               IN CHAR8 *           FieldName,
                               IN CHAR8 *           FieldValue
                             )
```

Set or update a HTTP header with the field name and corresponding value.

Parameters:

[in] *HttpIoHeader* Point to the HTTP header holder.

[in] *FieldName* Null terminated string which describes a field name.
[in] *FieldValue* Null terminated string which describes the corresponding field value.

Return values:

EFI_SUCCESS	The HTTP header has been set or updated.
EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_OUT_OF_RESOURCES	Insufficient resource to complete the operation.
Other	Unexpected error happened.

References HttpBootFindHeader().

Referenced by HttpBootGetBootFile().

VOID HttpBootShowIp4Addr (IN EFI_IPv4_ADDRESS * *Ip*)

This function is to display the IPv4 address.

Parameters:

[in] *Ip* The pointer to the IPv4 address.

Referenced by HttpBootDhcp4Dora().

VOID HttpBootUintnToAscDecWithFormat (IN UINTN *Number*,
IN UINT8 * *Buffer*,
IN INTN *Length*
)

This function is to convert UINTN to ASCII string with the required formatting.

Parameters:

[in] *Number* Numeric value to be converted.
[in] *Buffer* The pointer to the buffer for ASCII string.
[in] *Length* The length of the required format.

Referenced by HttpBootBuildDhcp4Options().

VOID EFIAPI HttpIoCommonNotify (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

Notify the callback function when an event is triggered.

Parameters:

[in] *Event* The triggered event.
[in] *Context* The opaque parameter to the function.

Referenced by HttpIoCreateIo().

EFI_STATUS HttpIoCreateIo (IN EFI_HANDLE *Image*,
IN EFI_HANDLE *Controller*,

```
                    IN UINT8                    IpVersion,
                    IN HTTP_IO_CONFIG_DATA *    ConfigData,
                    OUT HTTP_IO *               HttpIo
                  )
```

Create a HTTP_IO to access the HTTP service. It will create and configure a HTTP child handle.

Parameters:

[in]	*Image*	The handle of the driver image.
[in]	*Controller*	The handle of the controller.
[in]	*IpVersion*	IP_VERSION_4 or IP_VERSION_6.
[in]	*ConfigData*	The HTTP_IO configuration data.
[out]	*HttpIo*	The HTTP_IO.

Return values:

EFI_SUCCESS	The HTTP_IO is created and configured.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_UNSUPPORTED	One or more of the control options are not supported in the implementation.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
Others	Failed to create the HTTP_IO or configure it.

References HttpIoCommonNotify(), and HttpIoDestroyIo().

Referenced by HttpBootCreateHttpIo().

```
VOID HttpIoDestroyIo ( IN HTTP_IO *  HttpIo  )
```

Destroy the HTTP_IO and release the resouces.

Parameters:

[in]	*HttpIo*	The HTTP_IO which wraps the HTTP service to be destroyed.

Referenced by HttpBootDestroyIp4Children(), HttpBootStop(), and HttpIoCreateIo().

```
EFI_STATUS HttpIoRecvResponse ( IN HTTP_IO *                    HttpIo,
                                IN BOOLEAN                      RecvMsgHeader,
                                OUT HTTP_IO_RESOPNSE_DATA *     ResponseData
                              )
```

Synchronously receive a HTTP RESPONSE message from the server.

Parameters:

[in]	*HttpIo*	The HttpIo wrapping the HTTP service.
[in]	*RecvMsgHeader*	TRUE to receive a new HTTP response (from message header). FALSE to continue receive the previous response message.
[out]	*ResponseData*	Point to a wrapper of the received response data.

Return values:

EFI_SUCCESS	The HTTP resopnse is received.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.

EFI_DEVICE_ERROR	An unexpected network or system error occurred.
Others	Other errors as indicated.

Referenced by HttpBootGetBootFile().

```
EFI_STATUS HttpIoSendRequest ( IN HTTP_IO *                    HttpIo,
                               IN EFI_HTTP_REQUEST_DATA *      Request,
                               IN UINTN                        HeaderCount,
                               IN EFI_HTTP_HEADER *            Headers,
                               IN UINTN                        BodyLength,
                               IN VOID *                       Body
                             )
```

Synchronously send a HTTP REQUEST message to the server.

Parameters:

[in]	*HttpIo*	The HttpIo wrapping the HTTP service.
[in]	*Request*	A pointer to storage such data as URL and HTTP method.
[in]	*HeaderCount*	Number of HTTP header structures in Headers list.
[in]	*Headers*	Array containing list of HTTP headers.
[in]	*BodyLength*	Length in bytes of the HTTP body.
[in]	*Body*	Body associated with the HTTP request.

Return values:

EFI_SUCCESS	The HTTP request is trasmitted.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_DEVICE_ERROR	An unexpected network or system error occurred.
Others	Other errors as indicated.

Referenced by HttpBootGetBootFile().

NetworkPkg/HttpBootDxe/HttpBootSupport.h File Reference

Data Structures

struct HTTP_IO_HEADER
struct HTTP4_IO_CONFIG_DATA
union HTTP_IO_CONFIG_DATA
struct HTTP_IO
struct HTTP_IO_RESOPNSE_DATA

Functions

EFI_HANDLE HttpBootGetNicByIp4Children (IN EFI_HANDLE ControllerHandle)
VOID HttpBootUintnToAscDecWithFormat (IN UINTN Number, IN UINT8 *Buffer, IN INTN Length)
VOID HttpBootShowIp4Addr (IN EFI_IPv4_ADDRESS *Ip)
HTTP_IO_HEADER * HttpBootCreateHeader (IN UINTN MaxHeaderCount)
VOID HttpBootFreeHeader (IN HTTP_IO_HEADER *HttpIoHeader)
EFI_STATUS HttpBootSetHeader (IN HTTP_IO_HEADER *HttpIoHeader, IN CHAR8 *FieldName, IN CHAR8 *FieldValue)
EFI_STATUS HttpIoCreateIo (IN EFI_HANDLE Image, IN EFI_HANDLE Controller, IN UINT8 IpVersion, IN HTTP_IO_CONFIG_DATA *ConfigData, OUT HTTP_IO *HttpIo)
VOID HttpIoDestroyIo (IN HTTP_IO *HttpIo)
EFI_STATUS HttpIoSendRequest (IN HTTP_IO *HttpIo, IN EFI_HTTP_REQUEST_DATA *Request, IN UINTN HeaderCount, IN EFI_HTTP_HEADER *Headers, IN UINTN BodyLength, IN VOID *Body)
EFI_STATUS HttpIoRecvResponse (IN HTTP_IO *HttpIo, IN BOOLEAN RecvMsgHeader, OUT HTTP_IO_RESOPNSE_DATA *ResponseData)

Detailed Description

Support functions declaration for UEFI HTTP boot driver.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

HTTP_IO_HEADER* HttpBootCreateHeader (IN UINTN *MaxHeaderCount*)

Create a HTTP_IO_HEADER to hold the HTTP header items.

Parameters:

[in] *MaxHeaderCount* The maximun number of HTTP header in this holder.

Returns:

A pointer of the HTTP header holder or NULL if failed.

Referenced by HttpBootGetBootFile().

VOID HttpBootFreeHeader (IN HTTP_IO_HEADER * *HttpIoHeader*)

Destroy the HTTP_IO_HEADER and release the resouces.

Parameters:

[in] *HttpIoHeader* Point to the HTTP header holder to be destroyed.

Referenced by HttpBootGetBootFile().

EFI_HANDLE HttpBootGetNicByIp4Children (IN EFI_HANDLE *ControllerHandle*)

Get the Nic handle using any child handle in the IPv4 stack.

Parameters:

[in] *ControllerHandle* Pointer to child handle over IPv4.

Returns:

NicHandle The pointer to the Nic handle.

NULL Can't find the Nic handle.

Referenced by HttpBootDxeComponentNameGetControllerName(), and HttpBootIp4DxeDriverBindingStop().

```
EFI_STATUS HttpBootSetHeader ( IN HTTP_IO_HEADER *   HttpIoHeader,
                               IN CHAR8 *            FieldName,
                               IN CHAR8 *            FieldValue
                             )
```

Set or update a HTTP header with the field name and corresponding value.

Parameters:

[in] *HttpIoHeader* Point to the HTTP header holder.

[in] *FieldName* Null terminated string which describes a field name.

[in] *FieldValue* Null terminated string which describes the corresponding field value.

Return values:

EFI_SUCCESS	The HTTP header has been set or updated.
EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_OUT_OF_RESOURCES	Insufficient resource to complete the operation.
Other	Unexpected error happened.

References HttpBootFindHeader().

Referenced by HttpBootGetBootFile().

VOID HttpBootShowIp4Addr (IN EFI_IPv4_ADDRESS * *Ip*)

This function is to display the IPv4 address.

Parameters:

[in] *Ip* The pointer to the IPv4 address.

Referenced by HttpBootDhcp4Dora().

VOID HttpBootUintnToAscDecWithFormat (IN UINTN *Number*,
IN UINT8 * *Buffer*,
IN INTN *Length*
)

This function is to convert UINTN to ASCII string with the required formatting.

Parameters:

[in] *Number* Numeric value to be converted.
[in] *Buffer* The pointer to the buffer for ASCII string.
[in] *Length* The length of the required format.

Referenced by HttpBootBuildDhcp4Options().

EFI_STATUS HttpIoCreateIo (IN EFI_HANDLE *Image*,
IN EFI_HANDLE *Controller*,
IN UINT8 *IpVersion*,
IN HTTP_IO_CONFIG_DATA * *ConfigData*,
OUT HTTP_IO * *HttpIo*
)

Create a HTTP_IO to access the HTTP service. It will create and configure a HTTP child handle.

Parameters:

[in] *Image* The handle of the driver image.
[in] *Controller* The handle of the controller.
[in] *IpVersion* IP_VERSION_4 or IP_VERSION_6.
[in] *ConfigData* The HTTP_IO configuration data.
[out] *HttpIo* The HTTP_IO.

Return values:

EFI_SUCCESS The HTTP_IO is created and configured.
EFI_INVALID_PARAMETER One or more parameters are invalid.
EFI_UNSUPPORTED One or more of the control options are not supported in the implementation.
EFI_OUT_OF_RESOURCES Failed to allocate memory.
Others Failed to create the HTTP_IO or configure it.

References HttpIoCommonNotify(), and HttpIoDestroyIo().

Referenced by HttpBootCreateHttpIo().

VOID HttpIoDestroyIo (IN HTTP_IO * *HttpIo*)

Destroy the HTTP_IO and release the resouces.

Parameters:

[in] *HttpIo* The HTTP_IO which wraps the HTTP service to be destroyed.

Referenced by HttpBootDestroyIp4Children(), HttpBootStop(), and HttpIoCreateIo().

EFI_STATUS HttpIoRecvResponse (IN HTTP_IO * *HttpIo*,
IN BOOLEAN *RecvMsgHeader*,
OUT HTTP_IO_RESOPNSE_DATA * *ResponseData*
)

Synchronously receive a HTTP RESPONSE message from the server.

Parameters:

[in]	*HttpIo*	The HttpIo wrapping the HTTP service.
[in]	*RecvMsgHeader*	TRUE to receive a new HTTP response (from message header). FALSE to continue receive the previous response message.
[out]	*ResponseData*	Point to a wrapper of the received response data.

Return values:

EFI_SUCCESS	The HTTP resopnse is received.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_DEVICE_ERROR	An unexpected network or system error occurred.
Others	Other errors as indicated.

Referenced by HttpBootGetBootFile().

EFI_STATUS HttpIoSendRequest (IN HTTP_IO * *HttpIo*,
IN EFI_HTTP_REQUEST_DATA * *Request*,
IN UINTN *HeaderCount*,
IN EFI_HTTP_HEADER * *Headers*,
IN UINTN *BodyLength*,
IN VOID * *Body*
)

Synchronously send a HTTP REQUEST message to the server.

Parameters:

[in]	*HttpIo*	The HttpIo wrapping the HTTP service.
[in]	*Request*	A pointer to storage such data as URL and HTTP method.
[in]	*HeaderCount*	Number of HTTP header structures in Headers list.
[in]	*Headers*	Array containing list of HTTP headers.
[in]	*BodyLength*	Length in bytes of the HTTP body.
[in]	*Body*	Body associated with the HTTP request.

Return values:

EFI_SUCCESS	The HTTP request is trasmitted.
EFI_INVALID_PARAMETER	One or more parameters are invalid.

EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_DEVICE_ERROR	An unexpected network or system error occurred.
Others	Other errors as indicated.

Referenced by HttpBootGetBootFile().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/HttpDxe/ComponentName.c File Reference

Functions

EFI_STATUS EFIAPI HttpDxeComponentNameGetDriverName (IN EFI_COMPONENT_NAME2_PROTOCOL *This, IN CHAR8 *Language, OUT CHAR16 **DriverName)

EFI_STATUS EFIAPI HttpDxeComponentNameGetControllerName (IN EFI_COMPONENT_NAME2_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE ChildHandle, IN CHAR8 *Language, OUT CHAR16 **ControllerName)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gHttpDxeComponentName

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gHttpDxeComponentName2

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mHttpDxeDriverNameTable []

Detailed Description

Implementation of EFI_COMPONENT_NAME_PROTOCOL and EFI_COMPONENT_NAME2_PROTOCOL protocol.

Copyright (c) 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI HttpDxeComponentNameGetControllerName	(IN EFI_COMPONENT_NAME2_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_HANDLE	*ChildHandle*,
	IN CHAR8 *	*Language*,
	OUT CHAR16 **	*ControllerName*

)

Retrieves a Unicode string that is the user readable name of the controller that is being managed by an EFI Driver.

Parameters:

This	A pointer to the EFI_COMPONENT_NAME_PROTOCOL instance.
ControllerHandle	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
ChildHandle	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
Language	A pointer to a three character ISO 639-2 language identifier. This is the language of the controller name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer.
ControllerName	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language, from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

```
EFI_STATUS EFIAPI
HttpDxeComponentNameGetDriverName
  (  IN EFI_COMPONENT_NAME2_PROTOCOL *   This,
     IN CHAR8 *                          Language,
     OUT CHAR16 **                       DriverName
  )
```

Retrieves a Unicode string that is the user-readable name of the EFI Driver.

Parameters:

This	A pointer to the EFI_COMPONENT_NAME_PROTOCOL instance.
Language	A pointer to a three-character ISO 639-2 language identifier. This is the language of the driver name that that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer.
DriverName	A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by This in the language specified by Language.

Return values:

EFI_SUCCESS	The Unicode string for the Driver specified by This and the language specified by Language was returned in DriverName.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	DriverName is NULL.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gHttpDxeComponentName2, and mHttpDxeDriverNameTable.

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gHttpDxeComponentName

Initial value:

```
{
 (EFI_COMPONENT_NAME_GET_DRIVER_NAME)     HttpDxeComponentNameGetDriverName,
 (EFI_COMPONENT_NAME_GET_CONTROLLER_NAME) HttpDxeComponentNameGetControllerName,
 "eng"
}
```

Component Name Protocol instance

Referenced by HttpDxeDriverEntryPoint().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gHttpDxeComponentName2

Initial value:

```
{
 HttpDxeComponentNameGetDriverName,
 HttpDxeComponentNameGetControllerName,
 "en"
}
```

Component Name 2 Protocol instance

Referenced by HttpDxeComponentNameGetDriverName(), and HttpDxeDriverEntryPoint().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mHttpDxeDriverNameTable[]

Initial value:

```
{
 { "eng;en", (CHAR16 *) L"HttpDxe" },
 { NULL, NULL }
}
```

Table of driver names

Referenced by HttpDxeComponentNameGetDriverName().

- Files

- File List
- Globals

NetworkPkg/HttpDxe/ComponentName.h File Reference

Functions

EFI_STATUS EFIAPI HttpDxeComponentNameGetDriverName (IN EFI_COMPONENT_NAME2_PROTOCOL *This, IN CHAR8 *Language, OUT CHAR16 **DriverName)

EFI_STATUS EFIAPI HttpDxeComponentNameGetControllerName (IN EFI_COMPONENT_NAME2_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE ChildHandle, IN CHAR8 *Language, OUT CHAR16 **ControllerName)

Detailed Description

Header file for implementation of UEFI Component Name(2) protocol.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI
HttpDxeComponentNameGetControllerName (
    IN EFI_COMPONENT_NAME2_PROTOCOL *  This,
    IN EFI_HANDLE                      ControllerHandle,
    IN EFI_HANDLE                      ChildHandle,
    IN CHAR8 *                         Language,
    OUT CHAR16 **                      ControllerName
)
```

Retrieves a Unicode string that is the user readable name of the controller that is being managed by an EFI Driver.

Parameters:

This	A pointer to the EFI_COMPONENT_NAME_PROTOCOL instance.
ControllerHandle	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
ChildHandle	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.

Language	A pointer to a three character ISO 639-2 language identifier. This is the language of the controller name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer.
ControllerName	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language, from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

```
EFI_STATUS EFIAPI
HttpDxeComponentNameGetDriverName
  (
    IN EFI_COMPONENT_NAME2_PROTOCOL *   This,
    IN CHAR8 *                          Language,
    OUT CHAR16 **                       DriverName
  )
```

Retrieves a Unicode string that is the user-readable name of the EFI Driver.

Parameters:

This	A pointer to the EFI_COMPONENT_NAME_PROTOCOL instance.
Language	A pointer to a three-character ISO 639-2 language identifier. This is the language of the driver name that that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer.
DriverName	A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by This in the language specified by Language.

Return values:

EFI_SUCCESS	The Unicode string for the Driver specified by This and the language specified by Language was returned in DriverName.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	DriverName is NULL.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gHttpDxeComponentName2, and mHttpDxeDriverNameTable.

- Files

- File List
- Globals

NetworkPkg/HttpDxe/HttpDns.c File Reference

Functions

EFI_STATUS HttpDns4 (IN HTTP_PROTOCOL *HttpInstance, IN CHAR16 *HostName, OUT EFI_IPv4_ADDRESS *IpAddress)

Detailed Description

Routines for HttpDxe driver to perform DNS resolution based on UEFI DNS protocols.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS HttpDns4 ( IN HTTP_PROTOCOL *        HttpInstance,
                      IN CHAR16 *               HostName,
                      OUT EFI_IPv4_ADDRESS *    IpAddress
                    )
```

Retrieve the host address using the EFI_DNS4_PROTOCOL.

Parameters:

[in]	*HttpInstance*	Pointer to HTTP_PROTOCOL instance.
[in]	*HostName*	Pointer to buffer containing hostname.
[out]	*IpAddress*	On output, pointer to buffer containing IPv4 address.

Return values:

EFI_SUCCESS	Operation succeeded.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_DEVICE_ERROR	An unexpected network error occurred.
Others	Other errors as indicated.

References _HTTP_SERVICE::ControllerHandle, HttpCommonNotify(), and _HTTP_SERVICE::ImageHandle.

Referenced by EfiHttpRequest().

- File List
- Globals

NetworkPkg/HttpDxe/HttpDns.h File Reference

Functions

EFI_STATUS HttpDns4 (IN HTTP_PROTOCOL *HttpInstance, IN CHAR16 *HostName, OUT EFI_IPv4_ADDRESS *IpAddress)

Detailed Description

The header file of routines for HttpDxe driver to perform DNS resolution based on UEFI DNS protocols.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS HttpDns4 ( IN HTTP_PROTOCOL *        HttpInstance,
                      IN CHAR16 *               HostName,
                      OUT EFI_IPv4_ADDRESS *    IpAddress
                    )
```

Retrieve the host address using the EFI_DNS4_PROTOCOL.

Parameters:

[in]	*HttpInstance*	Pointer to HTTP_PROTOCOL instance.
[in]	*HostName*	Pointer to buffer containing hostname.
[out]	*IpAddress*	On output, pointer to buffer containing IPv4 address.

Return values:

EFI_SUCCESS	Operation succeeded.
EFI_OUT_OF_RESOURCES	Failed to allocate needed resources.
EFI_DEVICE_ERROR	An unexpected network error occurred.
Others	Other errors as indicated.

References _HTTP_SERVICE::ControllerHandle, HttpCommonNotify(), and _HTTP_SERVICE::ImageHandle.

Referenced by EfiHttpRequest().

- File List
- Globals

NetworkPkg/HttpDxe/HttpDriver.c File Reference

Functions

EFI_STATUS HttpCreateService (IN EFI_HANDLE Controller, IN EFI_HANDLE ImageHandle, OUT HTTP_SERVICE **ServiceData)

VOID HttpCleanService (IN HTTP_SERVICE *HttpService)

VOID EFIAPI HttpUtilitiesInstalledCallback (IN EFI_EVENT Event, IN VOID *Context)

EFI_STATUS EFIAPI HttpDxeDriverEntryPoint (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)

EFI_STATUS EFIAPI HttpDestroyChildEntryInHandleBuffer (IN LIST_ENTRY *Entry, IN VOID *Context)

EFI_STATUS EFIAPI HttpDxeDriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI HttpDxeDriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI HttpDxeDriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)

EFI_STATUS EFIAPI HttpServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN OUT EFI_HANDLE *ChildHandle)

EFI_STATUS EFIAPI HttpServiceBindingDestroyChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE ChildHandle)

Variables

EFI_HTTP_UTILITIES_PROTOCOL * mHttpUtilities = NULL

EFI_DRIVER_BINDING_PROTOCOL gHttpDxeDriverBinding

Detailed Description

The driver binding and service binding protocol for HttpDxe driver.

Copyright (c) 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

VOID HttpCleanService (IN HTTP_SERVICE * *HttpService*)

Release all the resource used the HTTP service binding instance.

Parameters:

HttpService The HTTP private instance.

Referenced by HttpDxeDriverBindingStart(), and HttpDxeDriverBindingStop().

EFI_STATUS HttpCreateService (IN EFI_HANDLE *Controller*,
IN EFI_HANDLE *ImageHandle*,
OUT HTTP_SERVICE ** *ServiceData*
)

Create a HTTP driver service binding private instance.

Parameters:

[in]	*Controller*	The controller that has TCP4 service binding installed.
[in]	*ImageHandle*	The HTTP driver's image handle.
[out]	*ServiceData*	Point to HTTP driver private instance.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate some resources.
EFI_SUCCESS	A new HTTP driver private instance is created.

References _HTTP_SERVICE::ChildrenList, _HTTP_SERVICE::ChildrenNumber, _HTTP_SERVICE::ControllerHandle, HTTP_SERVICE_SIGNATURE, HttpServiceBindingCreateChild(), HttpServiceBindingDestroyChild(), _HTTP_SERVICE::ImageHandle, _HTTP_SERVICE::ServiceBinding, and _HTTP_SERVICE::Signature.

Referenced by HttpDxeDriverBindingStart().

EFI_STATUS EFIAPI HttpDestroyChildEntryInHandleBuffer (IN LIST_ENTRY * *Entry*,
IN VOID * *Context*
)

Callback function which provided by user to remove one node in NetDestroyLinkList process.

Parameters:

[in]	*Entry*	The entry to be removed.
[in]	*Context*	Pointer to the callback context corresponds to the Context in NetDestroyLinkList.

Return values:

EFI_INVALID_PARAMETER	Any input parameter is NULL.
EFI_SUCCESS	The entry has been removed successfully.
Others	Fail to remove the entry.

References _HTTP_PROTOCOL::Handle, and HTTP_PROTOCOL_SIGNATURE.

Referenced by HttpDxeDriverBindingStop().

EFI_STATUS EFIAPI HttpDxeDriverBindingStart	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*
	)	

Starts a device controller or a bus controller.

The Start() function is designed to be invoked from the EFI boot service ConnectController(). As a result, much of the error checking on the parameters to Start() has been moved into this common boot service. It is legal to call Start() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE. 2. If RemainingDevicePath is not NULL, then it must be a pointer to a naturally aligned EFI_DEVICE_PATH_PROTOCOL. 3. Prior to calling Start(), the Supported() function for the driver specified by This must have been called with the same calling parameters, and Supported() must have returned EFI_SUCCESS.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to start. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For a bus driver, if this parameter is NULL, then handles for all the children of Controller are created by this driver. If this parameter is not NULL and the first Device Path Node is not the End of Device Path Node, then only the handle for the child device specified by the first Device Path Node of RemainingDevicePath is created by this driver. If the first Device Path Node of RemainingDevicePath is the End of Device Path Node, no child handle is created by this driver.

Return values:

EFI_SUCCESS	The device was started.
EFI_ALREADY_STARTED	This device is already running on ControllerHandle.
EFI_DEVICE_ERROR	The device could not be started due to a device error.Currently not implemented.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
Others	The driver failded to start the device.

References HttpCleanService(), HttpCreateService(), _HTTP_SERVICE::ServiceBinding, and _HTTP_SERVICE::TcpChildHandle.

EFI_STATUS EFIAPI HttpDxeDriverBindingStop	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN UINTN	*NumberOfChildren*,
	IN EFI_HANDLE *	*ChildHandleBuffer*
	)	

Stops a device controller or a bus controller.

The Stop() function is designed to be invoked from the EFI boot service DisconnectController(). As a result, much of the error checking on the parameters to Stop() has been moved into this common boot service. It is legal to call Stop() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE that was used on a previous call to this same driver's Start() function. 2. The first NumberOfChildren handles of ChildHandleBuffer must all be a valid EFI_HANDLE. In addition, all of these handles must have been created in this driver's Start() function, and the Start() function must have called OpenProtocol() on ControllerHandle with an Attribute of EFI_OPEN_PROTOCOL_BY_CHILD_CONTROLLER.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
[in]	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
[in]	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References HTTP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer, _HTTP_SERVICE::ChildrenList, HTTP_SERVICE_FROM_PROTOCOL, HttpCleanService(), HttpDestroyChildEntryInHandleBuffer(), HTTP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren, and HTTP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding.

```
EFI_STATUS EFIAPI
HttpDxeDriverBindingSupported
  (  IN EFI_DRIVER_BINDING_PROTOCOL *   This,
     IN EFI_HANDLE                      ControllerHandle,
     IN EFI_DEVICE_PATH_PROTOCOL *      RemainingDevicePath
  )
```

Tests to see if this driver supports a given controller. If a child device is provided, it further tests to see if this driver supports creating a handle for the specified child device.

This function checks to see if the driver specified by This supports the device specified by ControllerHandle. Drivers will typically use the device path attached to ControllerHandle and/or the services from the bus I/O abstraction attached to ControllerHandle to determine if the driver supports ControllerHandle. This function may be called many times during platform initialization. In order to reduce boot times, the tests performed by this function must be very small, and take as little time as possible to execute. This function must not change the state of any hardware devices, and this function must be aware that the device specified by ControllerHandle may already be managed by the same driver or a different driver. This function must match its calls to AllocatePages() with FreePages(), AllocatePool() with FreePool(), and OpenProtocol() with CloseProtocol(). Because ControllerHandle may have been previously started by the same driver, if a protocol is already in the opened state, then it must not be closed with CloseProtocol(). This is required to guarantee the state of ControllerHandle is not modified by this function.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.

`[in]` *RemainingDevicePath* A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by the driver specified by This.
EFI_ACCESS_DENIED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by a different driver or an application that requires exclusive access. Currently not implemented.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

```
EFI_STATUS EFIAPI HttpDxeDriverEntryPoint ( IN EFI_HANDLE          ImageHandle,
                                            IN EFI_SYSTEM_TABLE *  SystemTable
                                          )
```

This is the declaration of an EFI image entry point. This entry point is the same for UEFI Applications, UEFI OS Loaders, and UEFI Drivers including both device drivers and bus drivers.

Parameters:

ImageHandle	The firmware allocated handle for the UEFI image.
SystemTable	A pointer to the EFI System Table.

Return values:

EFI_SUCCESS	The operation completed successfully.
Others	An unexpected error occurred.

References gHttpDxeComponentName, gHttpDxeComponentName2, gHttpDxeDriverBinding, HttpUtilitiesInstalledCallback(), and mHttpUtilities.

```
EFI_STATUS EFIAPI                    IN
HttpServiceBindingCreateChild      ( EFI_SERVICE_BINDING_PROTOCOL  This,
                                     *
                                     IN OUT EFI_HANDLE *           ChildHandle
                                   )
```

Creates a child handle and installs a protocol.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

This	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
ChildHandle	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.
EFI_INVALID_PARAMETER	This is NULL, or ChildHandle is NULL.
EFI_OUT_OF_RESOURCES	There are not enough resources availabe to create the child.
other	The child handle was not created.

References _HTTP_SERVICE::ChildrenList, _HTTP_SERVICE::ChildrenNumber, gHttpDxeDriverBinding, _HTTP_PROTOCOL::Handle, _HTTP_PROTOCOL::Http, HTTP_SERVICE_FROM_PROTOCOL, HttpCleanProtocol(), HttpInitProtocol(), _HTTP_PROTOCOL::Link, and _HTTP_SERVICE::TcpChildHandle.

Referenced by HttpCreateService().

```
EFI_STATUS EFIAPI
HttpServiceBindingDestroyChild ( IN EFI_SERVICE_BINDING_PROTOCOL *  This,
                                 IN EFI_HANDLE                      ChildHandle
                               )
```

Destroys a child handle with a protocol installed on it.

The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

This	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
ChildHandle	Handle of the child to destroy

Return values:

EFI_SUCCES	The protocol was removed from ChildHandle.
EFI_UNSUPPORTED	ChildHandle does not support the protocol that is being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
other	The child handle was not destroyed

References _HTTP_SERVICE::ChildrenNumber, gHttpDxeDriverBinding, HTTP_INSTANCE_FROM_PROTOCOL, HTTP_SERVICE_FROM_PROTOCOL, HttpCleanProtocol(), _HTTP_PROTOCOL::InDestroy, _HTTP_PROTOCOL::Link, _HTTP_PROTOCOL::Service, and _HTTP_SERVICE::TcpChildHandle.

Referenced by HttpCreateService().

```
VOID EFIAPI HttpUtilitiesInstalledCallback ( IN EFI_EVENT  Event,
                                             IN VOID *     Context
                                           )
```

The event process routine when the http utilities protocol is installed in the system.

Parameters:

`[in]` *Event*	Not used.
`[in]` *Context*	The pointer to the IP4 config2 instance data.

References mHttpUtilities.

Referenced by HttpDxeDriverEntryPoint().

Variable Documentation

EFI_DRIVER_BINDING_PROTOCOL gHttpDxeDriverBinding

Initial value:

```
{
  HttpDxeDriverBindingSupported,
  HttpDxeDriverBindingStart,
  HttpDxeDriverBindingStop,
  HTTP_DRIVER_VERSION,
  NULL,
  NULL
}
```

Driver Binding Protocol instance

Referenced by HttpDxeDriverEntryPoint(), HttpServiceBindingCreateChild(), and HttpServiceBindingDestroyChild().

EFI_HTTP_UTILITIES_PROTOCOL* mHttpUtilities = NULL

Referenced by HttpDxeDriverEntryPoint(), HttpGenRequestString(), HttpResponseWorker(), and HttpUtilitiesInstalledCallback().

NetworkPkg/HttpDxe/HttpDriver.h File Reference

Data Structures

struct HTTP_DESTROY_CHILD_IN_HANDLE_BU

Defines

"memItemRight" valign="bottom">HTTP_DRIVER_VERSION 0xa

Functions

EFI_STATUS EFIAPI HttpDxeDriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *Remainin

EFI_STATUS EFIAPI HttpDxeDriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *Remainin

EFI_STATUS EFIAPI HttpDxeDriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildH

EFI_STATUS EFIAPI HttpServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL *This EFI_HANDLE *ChildHandle)

EFI_STATUS EFIAPI HttpServiceBindingDestroyChild (IN EFI_SERVICE_BINDING_PROTOCOL *This EFI_HANDLE ChildHandle)

Variables

EFI_DRIVER_BINDING_PROTOCOL gHttpDxeDriverBinding

EFI_COMPONENT_NAME2_PROTOCOL gHttpDxeComponentName2

EFI_COMPONENT_NAME_PROTOCOL gHttpDxeComponentName

EFI_HTTP_UTILITIES_PROTOCOL * mHttpUtilities

EFI_HTTP_PROTOCOL mEfiHttpProtocolTemplete

Detailed Description

The header files of the driver binding and service binding protocol for HttpDxe driver.

Copyright (c) 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Function Documentation

EFI_STATUS EFIAPI HttpDxeDriverBindingStart	(	IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
		IN EFI_HANDLE	*ControllerHandle*,
		IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*
	)		

Starts a device controller or a bus controller.

The Start() function is designed to be invoked from the EFI boot service ConnectController(). As a result, much of the error checking on the parameters to Start() has been moved into this common boot service. It is legal to call Start() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE. 2. If RemainingDevicePath is not NULL, then it must be a pointer to a naturally aligned EFI_DEVICE_PATH_PROTOCOL. 3. Prior to calling Start(), the Supported() function for the driver specified by This must have been called with the same calling parameters, and Supported() must have returned EFI_SUCCESS.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to start. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For a bus driver, if this parameter is NULL, then handles for all the children of Controller are created by this driver. If this parameter is not NULL and the first Device Path Node is not the End of Device Path Node, then only the handle for the child device specified by the first Device Path Node of RemainingDevicePath is created by this driver. If the first Device Path Node of RemainingDevicePath is the End of Device Path Node, no child handle is created by this driver.

Return values:

EFI_SUCCESS	The device was started.
EFI_DEVICE_ERROR	The device could not be started due to a device error.Currently not implemented.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
Others	The driver failded to start the device.

Starts a device controller or a bus controller.

The Start() function is designed to be invoked from the EFI boot service ConnectController(). As a result,

much of the error checking on the parameters to Start() has been moved into this common boot service. It is legal to call Start() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE. 2. If RemainingDevicePath is not NULL, then it must be a pointer to a naturally aligned EFI_DEVICE_PATH_PROTOCOL. 3. Prior to calling Start(), the Supported() function for the driver specified by This must have been called with the same calling parameters, and Supported() must have returned EFI_SUCCESS.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to start. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For a bus driver, if this parameter is NULL, then handles for all the children of Controller are created by this driver. If this parameter is not NULL and the first Device Path Node is not the End of Device Path Node, then only the handle for the child device specified by the first Device Path Node of RemainingDevicePath is created by this driver. If the first Device Path Node of RemainingDevicePath is the End of Device Path Node, no child handle is created by this driver.

Return values:

EFI_SUCCESS	The device was started.
EFI_ALREADY_STARTED	This device is already running on ControllerHandle.
EFI_DEVICE_ERROR	The device could not be started due to a device error.Currently not implemented.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
Others	The driver failded to start the device.

References HttpCleanService(), HttpCreateService(), _HTTP_SERVICE::ServiceBinding, and _HTTP_SERVICE::TcpChildHandle.

```
EFI_STATUS EFIAPI
HttpDxeDriverBindingStop
(
  IN EFI_DRIVER_BINDING_PROTOCOL *   This,
  IN EFI_HANDLE                      ControllerHandle,
  IN UINTN                           NumberOfChildren,
  IN EFI_HANDLE *                    ChildHandleBuffer
)
```

Stops a device controller or a bus controller.

The Stop() function is designed to be invoked from the EFI boot service DisconnectController(). As a result, much of the error checking on the parameters to Stop() has been moved into this common boot service. It is legal to call Stop() from other locations, but the following calling restrictions must be followed, or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE that was used on a previous call to this same driver's Start() function. 2. The first NumberOfChildren handles of ChildHandleBuffer must all be a valid EFI_HANDLE. In addition, all of these handles must have been created in this driver's Start() function, and the Start() function must have called OpenProtocol() on ControllerHandle with an Attribute of EFI_OPEN_PROTOCOL_BY_CHILD_CONTROLLER.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.

[in]	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
[in]	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
[in]	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References HTTP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer, _HTTP_SERVICE::ChildrenList, HTTP_SERVICE_FROM_PROTOCOL, HttpCleanService(), HttpDestroyChildEntryInHandleBuffer(), HTTP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren, and HTTP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding.

```
EFI_STATUS EFIAPI
HttpDxeDriverBindingSupported
  (
    IN EFI_DRIVER_BINDING_PROTOCOL *   This,
    IN EFI_HANDLE                      ControllerHandle,
    IN EFI_DEVICE_PATH_PROTOCOL *      RemainingDevicePath
  )
```

Tests to see if this driver supports a given controller. If a child device is provided, it further tests to see if this driver supports creating a handle for the specified child device.

This function checks to see if the driver specified by This supports the device specified by ControllerHandle. Drivers will typically use the device path attached to ControllerHandle and/or the services from the bus I/O abstraction attached to ControllerHandle to determine if the driver supports ControllerHandle. This function may be called many times during platform initialization. In order to reduce boot times, the tests performed by this function must be very small, and take as little time as possible to execute. This function must not change the state of any hardware devices, and this function must be aware that the device specified by ControllerHandle may already be managed by the same driver or a different driver. This function must match its calls to AllocatePages() with FreePages(), AllocatePool() with FreePool(), and OpenProtocol() with CloseProtocol(). Because ControllerHandle may have been previously started by the same driver, if a protocol is already in the opened state, then it must not be closed with CloseProtocol(). This is required to guarantee the state of ControllerHandle is not modified by this function.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	

	The device specified by ControllerHandle and RemainingDevicePath is already being managed by the driver specified by This.
EFI_ACCESS_DENIED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by a different driver or an application that requires exclusive access. Currently not implemented.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

```
EFI_STATUS EFIAPI
HttpServiceBindingCreateChild
(
  IN EFI_SERVICE_BINDING_PROTOCOL *   This,
  IN OUT EFI_HANDLE *                 ChildHandle
)
```

Creates a child handle and installs a protocol.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

This	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
ChildHandle	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.
EFI_INVALID_PARAMETER	This is NULL, or ChildHandle is NULL.
EFI_OUT_OF_RESOURCES	There are not enough resources availabe to create the child.
other	The child handle was not created.

References _HTTP_SERVICE::ChildrenList, _HTTP_SERVICE::ChildrenNumber, gHttpDxeDriverBinding, _HTTP_PROTOCOL::Handle, _HTTP_PROTOCOL::Http, HTTP_SERVICE_FROM_PROTOCOL, HttpCleanProtocol(), HttpInitProtocol(), _HTTP_PROTOCOL::Link, and _HTTP_SERVICE::TcpChildHandle.

Referenced by HttpCreateService().

```
EFI_STATUS EFIAPI
HttpServiceBindingDestroyChild
(
  IN EFI_SERVICE_BINDING_PROTOCOL *   This,
  IN EFI_HANDLE                       ChildHandle
)
```

Destroys a child handle with a protocol installed on it.

The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

This	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.

ChildHandle Handle of the child to destroy

Return values:

EFI_SUCCES	The protocol was removed from ChildHandle.
EFI_UNSUPPORTED	ChildHandle does not support the protocol that is being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
other	The child handle was not destroyed

References _HTTP_SERVICE::ChildrenNumber, gHttpDxeDriverBinding, HTTP_INSTANCE_FROM_PROTOCOL, HTTP_SERVICE_FROM_PROTOCOL, HttpCleanProtocol(), _HTTP_PROTOCOL::InDestroy, _HTTP_PROTOCOL::Link, _HTTP_PROTOCOL::Service, and _HTTP_SERVICE::TcpChildHandle.

Referenced by HttpCreateService().

Variable Documentation

EFI_COMPONENT_NAME_PROTOCOL gHttpDxeComponentName
Component Name Protocol instance

Referenced by HttpDxeDriverEntryPoint().

EFI_COMPONENT_NAME2_PROTOCOL gHttpDxeComponentName2
Component Name 2 Protocol instance

Referenced by HttpDxeComponentNameGetDriverName(), and HttpDxeDriverEntryPoint().

EFI_DRIVER_BINDING_PROTOCOL gHttpDxeDriverBinding
Driver Binding Protocol instance

Referenced by HttpDxeDriverEntryPoint(), HttpServiceBindingCreateChild(), and HttpServiceBindingDestroyChild().

EFI_HTTP_PROTOCOL mEfiHttpProtocolTemplete

EFI_HTTP_UTILITIES_PROTOCOL* mHttpUtilities

Referenced by HttpDxeDriverEntryPoint(), HttpGenRequestString(), HttpResponseWorker(), and HttpUtilitiesInstalledCallback().

NetworkPkg/HttpDxe/HttpImpl.c File Reference

Functions

EFI_STATUS EFIAPI EfiHttpGetModeData (IN EFI_HTTP_PROTOCOL *This, OUT EFI_HTTP_CONFIG_DATA *HttpConfigData)
EFI_STATUS EFIAPI EfiHttpConfigure (IN EFI_HTTP_PROTOCOL *This, IN EFI_HTTP_CONFIG_DATA *HttpConfigData)
EFI_STATUS EFIAPI EfiHttpRequest (IN EFI_HTTP_PROTOCOL *This, IN EFI_HTTP_TOKEN *Token)
EFI_STATUS EFIAPI HttpCancelTokens (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Context)
EFI_STATUS HttpCancel (IN HTTP_PROTOCOL *HttpInstance, IN EFI_HTTP_TOKEN *Token)
EFI_STATUS EFIAPI EfiHttpCancel (IN EFI_HTTP_PROTOCOL *This, IN EFI_HTTP_TOKEN *Token)
EFI_STATUS EFIAPI HttpBodyParserCallback (IN HTTP_BODY_PARSE_EVENT EventType, IN CHAR8 *Data, IN UINTN Length, IN VOID *Context)
EFI_STATUS HttpResponseWorker (IN HTTP_TOKEN_WRAP *Wrap)
EFI_STATUS EFIAPI EfiHttpResponse (IN EFI_HTTP_PROTOCOL *This, IN EFI_HTTP_TOKEN *Token)
EFI_STATUS EFIAPI EfiHttpPoll (IN EFI_HTTP_PROTOCOL *This)

Variables

EFI_HTTP_PROTOCOL mEfiHttpTemplate

Detailed Description

Implementation of EFI_HTTP_PROTOCOL protocol interfaces.

Copyright (c) 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI EfiHttpCancel (
    IN EFI_HTTP_PROTOCOL *  This,
    IN EFI_HTTP_TOKEN *     Token
)
```

Abort an asynchronous HTTP request or response token.

The Cancel() function aborts a pending HTTP request or response transaction. If Token is not NULL and the token is in transmit or receive queues when it is being cancelled, its Token->Status will be set to EFI_ABORTED and then Token->Event will be signaled. If the token is not in one of the queues, which usually means that the asynchronous operation has completed, EFI_NOT_FOUND is returned. If Token is NULL, all asynchronous tokens issued by Request() or Response() will be aborted.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_PROTOCOL instance.
[in]	*Token*	Point to storage containing HTTP request or response token.

Return values:

EFI_SUCCESS	Request and Response queues are successfully flushed.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This instance hasn't been configured.
EFI_NO_MAPPING	When using the default address, configuration (DHCP, BOOTP, RARP, etc.) hasn't finished yet.
EFI_NOT_FOUND	The asynchronous request or response token is not found.
EFI_UNSUPPORTED	The implementation does not support this function.

References HTTP_INSTANCE_FROM_PROTOCOL, HTTP_STATE_TCP_CONNECTED, HttpCancel(), and _HTTP_PROTOCOL::State.

Referenced by EfiHttpRequest().

```
EFI_STATUS EFIAPI EfiHttpConfigure ( IN EFI_HTTP_PROTOCOL *        This,
                                     IN EFI_HTTP_CONFIG_DATA *     HttpConfigData
                                   )
```

Initialize or brutally reset the operational parameters for this EFI HTTP instance.

The Configure() function does the following: When HttpConfigData is not NULL Initialize this EFI HTTP instance by configuring timeout, local address, port, etc. When HttpConfigData is NULL, reset this EFI HTTP instance by closing all active connections with remote hosts, canceling all asynchronous tokens, and flush request and response buffers without informing the appropriate hosts.

Except for GetModeData() and Configure(), No other EFI HTTP function can be executed by this instance until the Configure() function is executed and returns successfully.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_PROTOCOL instance.
[in]	*HttpConfigData*	Pointer to the configure data to configure the instance.

Return values:

EFI_SUCCESS	Operation succeeded.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. HttpConfigData->LocalAddressIsIPv6 is FALSE and HttpConfigData->IPv4Node is NULL. HttpConfigData->LocalAddressIsIPv6 is TRUE and HttpConfigData->IPv6Node is NULL.
EFI_ALREADY_STARTED	Reinitialize this HTTP instance without calling Configure() with NULL to reset it.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_OUT_OF_RESOURCES	

	Could not allocate enough system resources when executing Configure().
EFI_UNSUPPORTED	One or more options in HttpConfigData are not supported in the implementation.

References HTTP_INSTANCE_FROM_PROTOCOL, HTTP_STATE_HTTP_CONFIGED, HTTP_STATE_UNCONFIGED, HttpCleanProtocol(), HttpInitProtocol(), _HTTP_PROTOCOL::HttpVersion, _HTTP_PROTOCOL::IPv4Node, _HTTP_PROTOCOL::LocalAddressIsIPv6, _HTTP_PROTOCOL::Service, _HTTP_PROTOCOL::State, and _HTTP_PROTOCOL::TimeOutMillisec.

```
EFI_STATUS EFIAPI EfiHttpGetModeData ( IN EFI_HTTP_PROTOCOL *          This,
                                       OUT EFI_HTTP_CONFIG_DATA *      HttpConfigData
                                     )
```

Returns the operational parameters for the current HTTP child instance.

The GetModeData() function is used to read the current mode data (operational parameters) for this HTTP protocol instance.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_PROTOCOL instance.
[out]	*HttpConfigData*	Point to buffer for operational parameters of this HTTP instance.

Return values:

EFI_SUCCESS	Operation succeeded.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. HttpConfigData is NULL. HttpConfigData->AccessPoint is NULL.
EFI_NOT_STARTED	The HTTP instance is not configured.

References HTTP_INSTANCE_FROM_PROTOCOL, HTTP_STATE_HTTP_CONFIGED, _HTTP_PROTOCOL::HttpVersion, _HTTP_PROTOCOL::IPv4Node, _HTTP_PROTOCOL::LocalAddressIsIPv6, _HTTP_PROTOCOL::State, and _HTTP_PROTOCOL::TimeOutMillisec.

```
EFI_STATUS EFIAPI EfiHttpPoll ( IN EFI_HTTP_PROTOCOL * This )
```

The Poll() function can be used by network drivers and applications to increase the rate that data packets are moved between the communication devices and the transmit and receive queues.

In some systems, the periodic timer event in the managed network driver may not poll the underlying communications device fast enough to transmit and/or receive all data packets without missing incoming packets or dropping outgoing packets. Drivers and applications that are experiencing packet loss should try calling the Poll() function more often.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_PROTOCOL instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_READY	No incoming or outgoing data is processed.
EFI_NOT_STARTED	This EFI HTTP Protocol instance has not been started.

References HTTP_INSTANCE_FROM_PROTOCOL, HTTP_STATE_TCP_CONNECTED, _HTTP_PROTOCOL::LocalAddressIsIPv6, _HTTP_PROTOCOL::State, and _HTTP_PROTOCOL::Tcp4.

```
EFI_STATUS EFIAPI EfiHttpRequest ( IN EFI_HTTP_PROTOCOL *  This,
                                   IN EFI_HTTP_TOKEN *     Token
                                 )
```

The Request() function queues an HTTP request to this HTTP instance.

Similar to Transmit() function in the EFI TCP driver. When the HTTP request is sent successfully, or if there is an error, Status in token will be updated and Event will be signaled.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_PROTOCOL instance.
[in]	*Token*	Pointer to storage containing HTTP request token.

Return values:

EFI_SUCCESS	Outgoing data was processed.
EFI_NOT_STARTED	This EFI HTTP Protocol instance has not been started.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_TIMEOUT	Data was dropped out of the transmit or receive queue.
EFI_OUT_OF_RESOURCES	Could not allocate enough system resources.
EFI_UNSUPPORTED	The HTTP method is not supported in current implementation.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. Token->Message is NULL. Token->Message->Body is not NULL, Token->Message->BodyLength is non-zero, and Token->Message->Data is NULL, but a previous call to Request()has not been completed successfully.

References EfiHttpCancel(), HTTP_DEFAULT_PORT, HTTP_INSTANCE_FROM_PROTOCOL, HTTP_STATE_HTTP_CONFIGED, HTTP_URL_BUFFER_LEN, HttpCloseConnection(), HttpCloseTcp4ConnCloseEvent(), HttpConfigureTcp4(), HttpConnectTcp4(), HttpCreateTcp4TxEvent(), HttpDns4(), HttpGenRequestString(), HTTP_TOKEN_WRAP::HttpInstance, HttpTcpNotReady(), HTTP_TOKEN_WRAP::HttpToken, HttpTokenExist(), HttpTransmitTcp4(), _HTTP_PROTOCOL::LocalAddressIsIPv6, HTTP_TCP_TOKEN_WRAP::Method, _HTTP_PROTOCOL::RemoteAddr, _HTTP_PROTOCOL::RemoteHost, _HTTP_PROTOCOL::RemotePort, _HTTP_PROTOCOL::State, _HTTP_PROTOCOL::Tcp4, HTTP_TOKEN_WRAP::TcpWrap, HTTP_TCP_TOKEN_WRAP::TxToken, _HTTP_PROTOCOL::TxTokens, and _HTTP_PROTOCOL::Url.

```
EFI_STATUS EFIAPI EfiHttpResponse ( IN EFI_HTTP_PROTOCOL *  This,
                                    IN EFI_HTTP_TOKEN *     Token
                                  )
```

The Response() function queues an HTTP response to this HTTP instance, similar to Receive() function in the EFI TCP driver. When the HTTP request is sent successfully, or if there is an error, Status in token will be updated and Event will be signaled.

The HTTP driver will queue a receive token to the underlying TCP instance. When data is received in the underlying TCP instance, the data will be parsed and Token will be populated with the response data. If the data received from the remote host contains an incomplete or invalid HTTP header, the HTTP driver will continue waiting (asynchronously) for more data to be sent from the remote host before signaling Event in Token.

It is the responsibility of the caller to allocate a buffer for Body and specify the size in BodyLength. If the remote host provides a response that contains a content body, up to BodyLength bytes will be copied from the receive buffer into Body and BodyLength will be updated with the amount of bytes received and copied to Body. This allows the client to download a large file in chunks instead of into one contiguous block of memory. Similar to HTTP request, if Body is not NULL and BodyLength is non-zero and all other fields are NULL or 0, the HTTP driver will queue a receive token to underlying TCP instance. If data arrives in the receive buffer, up to BodyLength bytes of data will be copied to Body. The HTTP driver will then update BodyLength with the amount of bytes received and copied to Body.

If the HTTP driver does not have an open underlying TCP connection with the host specified in the response URL, Request() will return EFI_ACCESS_DENIED. This is consistent with RFC 2616 recommendation that HTTP clients should attempt to maintain an open TCP connection between client and host.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_PROTOCOL instance.
[in]	*Token*	Pointer to storage containing HTTP response token.

Return values:

EFI_SUCCESS	Allocation succeeded.
EFI_NOT_STARTED	This EFI HTTP Protocol instance has not been initialized.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. Token is NULL. Token->Message->Headers is NULL. Token->Message is NULL. Token->Message->Body is not NULL, Token->Message->BodyLength is non-zero, and Token->Message->Data is NULL, but a previous call to Response() has not been completed successfully.
EFI_OUT_OF_RESOURCES	Could not allocate enough system resources.
EFI_ACCESS_DENIED	An open TCP connection is not present with the host specified by response URL.

References HTTP_INSTANCE_FROM_PROTOCOL, HTTP_STATE_TCP_CONNECTED, HttpCreateTcp4RxEvent(), HTTP_TOKEN_WRAP::HttpInstance, HttpResponseWorker(), HTTP_TOKEN_WRAP::HttpToken, HttpTokenExist(), _HTTP_PROTOCOL::LocalAddressIsIPv6, HTTP_TCP_TOKEN_WRAP::RxToken, _HTTP_PROTOCOL::RxTokens, _HTTP_PROTOCOL::State, and HTTP_TOKEN_WRAP::TcpWrap.

```
EFI_STATUS EFIAPI HttpBodyParserCallback ( IN HTTP_BODY_PARSE_EVENT  EventType,
                                           IN CHAR8 *                Data,
                                           IN UINTN                  Length,
                                           IN VOID *                 Context
                                         )
```

A callback function to intercept events during message parser.

This function will be invoked during HttpParseMessageBody() with various events type. An error return status of the callback function will cause the HttpParseMessageBody() aborted.

Parameters:

[in]	*EventType*	Event type of this callback call.
[in]	*Data*	A pointer to data buffer.
[in]	*Length*	Length in bytes of the Data.
[in]	*Context*	Callback context set by HttpInitMsgParser().

Return values:

EFI_SUCCESS	Continue to parser the message body.

References HTTP_TOKEN_WRAP::HttpInstance, and _HTTP_PROTOCOL::NextMsg.

Referenced by HttpResponseWorker().

```
EFI_STATUS HttpCancel ( IN HTTP_PROTOCOL *   HttpInstance,
                        IN EFI_HTTP_TOKEN *  Token
                      )
```

Cancel the user's receive/transmit request. It is the worker function of EfiHttpCancel API. If a matching token is found, it will call HttpCancelTokens to cancel the token.

Parameters:

[in]	*HttpInstance*	Pointer to HTTP_PROTOCOL structure.
[in]	*Token*	The token to cancel. If NULL, all token will be cancelled.

Return values:

EFI_SUCCESS	The token is cancelled.
EFI_NOT_FOUND	The asynchronous request or response token is not found.
Others	Other error as indicated.

References HttpCancelTokens().

Referenced by EfiHttpCancel().

```
EFI_STATUS EFIAPI HttpCancelTokens ( IN NET_MAP *          Map,
                                     IN NET_MAP_ITEM *     Item,
                                     IN VOID *             Context
                                   )
```

Cancel a TxToken or RxToken.

Parameters:

[in]	*Map*	The HTTP instance's token queue.
[in]	*Item*	Object container for one HTTP token and token's wrap.
[in]	*Context*	The user's token to cancel.

Return values:

EFI_SUCCESS	Continue to check the next Item.
EFI_ABORTED	The user's Token (Token != NULL) is cancelled.

References HTTP_TCP_TOKEN_WRAP::RxToken, HTTP_TOKEN_WRAP::TcpWrap, and HTTP_TCP_TOKEN_WRAP::TxToken.

Referenced by HttpCancel().

EFI_STATUS HttpResponseWorker (IN HTTP_TOKEN_WRAP * *Wrap*)
The work function of EfiHttpResponse().

Parameters:

[in] *Wrap* Pointer to HTTP token's wrap data.

Return values:

EFI_SUCCESS	Allocation succeeded.
EFI_OUT_OF_RESOURCES	Failed to complete the opration due to lack of resources.
EFI_NOT_READY	Can't find a corresponding TxToken or the EFI_HTTP_UTILITIES_PROTOCOL is not available.

References _HTTP_PROTOCOL::CacheBody, _HTTP_PROTOCOL::CacheLen, _HTTP_PROTOCOL::CacheOffset, DEF_BUF_LEN, HTTP_CRLF_STR, HTTP_END_OF_HDR_STR, HTTP_VERSION_STR, HttpBodyParserCallback(), HttpCreateTcp4RxEventForHeader(), HttpMappingToStatusCode(), HTTP_TOKEN_WRAP::HttpToken, _HTTP_PROTOCOL::IsRxDone, mHttpUtilities, _HTTP_PROTOCOL::MsgParser, _HTTP_PROTOCOL::NextMsg, _HTTP_PROTOCOL::RxToken, _HTTP_PROTOCOL::Tcp4, and _HTTP_PROTOCOL::TxTokens.

Referenced by EfiHttpResponse(), and HttpTcpReceive().

Variable Documentation

EFI_HTTP_PROTOCOL mEfiHttpTemplate

Initial value:

```
{
 EfiHttpGetModeData,
 EfiHttpConfigure,
 EfiHttpRequest,
 EfiHttpCancel,
 EfiHttpResponse,
 EfiHttpPoll
}
```

Referenced by HttpInitProtocol().

NetworkPkg/HttpDxe/HttpImpl.h File Reference

Defines

"memItemRight" valign="bottom">HTTP_DEFAULT_PORT 80

#define HTTP_END_OF_HDR_STR "\r\n\r\n"

#define HTTP_CRLF_STR "\r\n"

#define HTTP_VERSION_STR "HTTP/1.1"

#define HTTP_VERSION_CRLF_STR " HTTP/1.1\r\n"

#define HTTP_GET_STR "GET "

#define HTTP_HEAD_STR "HEAD "

#define HTTP_MAXIMUM_METHOD_LEN sizeof ("CONNECT")

Functions

EFI_STATUS EFIAPI EfiHttpGetModeData (IN EFI_HTTP_PROTOCOL *This, OUT EFI_HTTP_CONFIG_DATA *HttpConfigData)

EFI_STATUS EFIAPI EfiHttpConfigure (IN EFI_HTTP_PROTOCOL *This, IN EFI_HTTP_CONFIG_DATA *HttpConfigData)

EFI_STATUS EFIAPI EfiHttpRequest (IN EFI_HTTP_PROTOCOL *This, IN EFI_HTTP_TOKEN *Token)

EFI_STATUS EFIAPI EfiHttpCancel (IN EFI_HTTP_PROTOCOL *This, IN EFI_HTTP_TOKEN *Token)

EFI_STATUS EFIAPI EfiHttpResponse (IN EFI_HTTP_PROTOCOL *This, IN EFI_HTTP_TOKEN *Token)

EFI_STATUS EFIAPI EfiHttpPoll (IN EFI_HTTP_PROTOCOL *This)

Variables

EFI_HTTP_PROTOCOL mEfiHttpTemplate

Detailed Description

The header files of implementation of EFI_HTTP_PROTOCOL protocol interfaces.

Copyright (c) 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at

http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by HttpResponseWorker().

#define HTTP_DEFAULT_PORT 80

Referenced by EfiHttpRequest().

#define HTTP_END_OF_HDR_STR "\r\n\r\n"

Referenced by HttpResponseWorker().

#define HTTP_GET_STR "GET "

Referenced by HttpGenRequestString().

#define HTTP_HEAD_STR "HEAD "

Referenced by HttpGenRequestString().

#define HTTP_MAXIMUM_METHOD_LEN sizeof ("CONNECT")

Referenced by HttpGenRequestString().

#define HTTP_VERSION_CRLF_STR " HTTP/1.1\r\n"

Referenced by HttpGenRequestString().

#define HTTP_VERSION_STR "HTTP/1.1"

Referenced by HttpResponseWorker().

Function Documentation

EFI_STATUS EFIAPI EfiHttpCancel (IN EFI_HTTP_PROTOCOL * *This*,

```
                                    IN EFI_HTTP_TOKEN *    Token
                                )
```

Abort an asynchronous HTTP request or response token.

The Cancel() function aborts a pending HTTP request or response transaction. If Token is not NULL and the token is in transmit or receive queues when it is being cancelled, its Token->Status will be set to EFI_ABORTED and then Token->Event will be signaled. If the token is not in one of the queues, which usually means that the asynchronous operation has completed, EFI_NOT_FOUND is returned. If Token is NULL, all asynchronous tokens issued by Request() or Response() will be aborted.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_PROTOCOL instance.
[in]	*Token*	Point to storage containing HTTP request or response token.

Return values:

EFI_SUCCESS	Request and Response queues are successfully flushed.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This instance hasn't been configured.
EFI_NO_MAPPING	When using the default address, configuration (DHCP, BOOTP, RARP, etc.) hasn't finished yet.
EFI_NOT_FOUND	The asynchronous request or response token is not found.
EFI_UNSUPPORTED	The implementation does not support this function.

References HTTP_INSTANCE_FROM_PROTOCOL, HTTP_STATE_TCP_CONNECTED, HttpCancel(), and _HTTP_PROTOCOL::State.

Referenced by EfiHttpRequest().

```
EFI_STATUS EFIAPI EfiHttpConfigure  ( IN EFI_HTTP_PROTOCOL *        This,
                                      IN EFI_HTTP_CONFIG_DATA *     HttpConfigData
                                    )
```

Initialize or brutally reset the operational parameters for this EFI HTTP instance.

The Configure() function does the following: When HttpConfigData is not NULL Initialize this EFI HTTP instance by configuring timeout, local address, port, etc. When HttpConfigData is NULL, reset this EFI HTTP instance by closing all active connections with remote hosts, canceling all asynchronous tokens, and flush request and response buffers without informing the appropriate hosts.

Except for GetModeData() and Configure(), No other EFI HTTP function can be executed by this instance until the Configure() function is executed and returns successfully.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_PROTOCOL instance.
[in]	*HttpConfigData*	Pointer to the configure data to configure the instance.

Return values:

EFI_SUCCESS	Operation succeeded.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. HttpConfigData->LocalAddressIsIPv6 is FALSE and HttpConfigData->IPv4Node is NULL. HttpConfigData->LocalAddressIsIPv6 is TRUE and HttpConfigData->IPv6Node is NULL.

EFI_ALREADY_STARTED	Reinitialize this HTTP instance without calling Configure() with NULL to reset it.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_OUT_OF_RESOURCES	Could not allocate enough system resources when executing Configure().
EFI_UNSUPPORTED	One or more options in ConfigData are not supported in the implementation.

Initialize or brutally reset the operational parameters for this EFI HTTP instance.

The Configure() function does the following: When HttpConfigData is not NULL Initialize this EFI HTTP instance by configuring timeout, local address, port, etc. When HttpConfigData is NULL, reset this EFI HTTP instance by closing all active connections with remote hosts, canceling all asynchronous tokens, and flush request and response buffers without informing the appropriate hosts.

Except for GetModeData() and Configure(), No other EFI HTTP function can be executed by this instance until the Configure() function is executed and returns successfully.

Parameters:

`[in]`	*This*	Pointer to EFI_HTTP_PROTOCOL instance.
`[in]`	*HttpConfigData*	Pointer to the configure data to configure the instance.

Return values:

EFI_SUCCESS	Operation succeeded.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. HttpConfigData->LocalAddressIsIPv6 is FALSE and HttpConfigData->IPv4Node is NULL. HttpConfigData->LocalAddressIsIPv6 is TRUE and HttpConfigData->IPv6Node is NULL.
EFI_ALREADY_STARTED	Reinitialize this HTTP instance without calling Configure() with NULL to reset it.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_OUT_OF_RESOURCES	Could not allocate enough system resources when executing Configure().
EFI_UNSUPPORTED	One or more options in HttpConfigData are not supported in the implementation.

References HTTP_INSTANCE_FROM_PROTOCOL, HTTP_STATE_HTTP_CONFIGED, HTTP_STATE_UNCONFIGED, HttpCleanProtocol(), HttpInitProtocol(), _HTTP_PROTOCOL::HttpVersion, _HTTP_PROTOCOL::IPv4Node, _HTTP_PROTOCOL::LocalAddressIsIPv6, _HTTP_PROTOCOL::Service, _HTTP_PROTOCOL::State, and _HTTP_PROTOCOL::TimeOutMillisec.

```
EFI_STATUS EFIAPI EfiHttpGetModeData ( IN EFI_HTTP_PROTOCOL *          This,
                                       OUT EFI_HTTP_CONFIG_DATA *      HttpConfigData
                                     )
```

Returns the operational parameters for the current HTTP child instance.

The GetModeData() function is used to read the current mode data (operational parameters) for this HTTP protocol instance.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_PROTOCOL instance.
[out]	*HttpConfigData*	Point to buffer for operational parameters of this HTTP instance.

Return values:

EFI_SUCCESS	Operation succeeded.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. HttpConfigData is NULL. HttpConfigData->AccessPoint is NULL.
EFI_NOT_STARTED	The HTTP instance is not configured.

References HTTP_INSTANCE_FROM_PROTOCOL, HTTP_STATE_HTTP_CONFIGED, _HTTP_PROTOCOL::HttpVersion, _HTTP_PROTOCOL::IPv4Node, _HTTP_PROTOCOL::LocalAddressIsIPv6, _HTTP_PROTOCOL::State, and _HTTP_PROTOCOL::TimeOutMillisec.

```
EFI_STATUS EFIAPI EfiHttpPoll ( IN EFI_HTTP_PROTOCOL * This )
```

The Poll() function can be used by network drivers and applications to increase the rate that data packets are moved between the communication devices and the transmit and receive queues.

In some systems, the periodic timer event in the managed network driver may not poll the underlying communications device fast enough to transmit and/or receive all data packets without missing incoming packets or dropping outgoing packets. Drivers and applications that are experiencing packet loss should try calling the Poll() function more often.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_PROTOCOL instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_READY	No incoming or outgoing data is processed.
EFI_NOT_STARTED	This EFI HTTP Protocol instance has not been started.

References HTTP_INSTANCE_FROM_PROTOCOL, HTTP_STATE_TCP_CONNECTED, _HTTP_PROTOCOL::LocalAddressIsIPv6, _HTTP_PROTOCOL::State, and _HTTP_PROTOCOL::Tcp4.

```
EFI_STATUS EFIAPI EfiHttpRequest ( IN EFI_HTTP_PROTOCOL * This,
                                   IN EFI_HTTP_TOKEN *    Token
                                 )
```

The Request() function queues an HTTP request to this HTTP instance.

Similar to Transmit() function in the EFI TCP driver. When the HTTP request is sent successfully, or if there is an error, Status in token will be updated and Event will be signaled.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_PROTOCOL instance.
[in]	*Token*	Pointer to storage containing HTTP request token.

Return values:

EFI_SUCCESS	Outgoing data was processed.
EFI_NOT_STARTED	This EFI HTTP Protocol instance has not been started.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_TIMEOUT	Data was dropped out of the transmit or receive queue.
EFI_OUT_OF_RESOURCES	Could not allocate enough system resources.
EFI_UNSUPPORTED	The HTTP method is not supported in current implementation.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. Token->Message is NULL. Token->Message->Body is not NULL, Token->Message->BodyLength is non-zero, and Token->Message->Data is NULL, but a previous call to Request()has not been completed successfully.

References EfiHttpCancel(), HTTP_DEFAULT_PORT, HTTP_INSTANCE_FROM_PROTOCOL, HTTP_STATE_HTTP_CONFIGED, HTTP_URL_BUFFER_LEN, HttpCloseConnection(), HttpCloseTcp4ConnCloseEvent(), HttpConfigureTcp4(), HttpConnectTcp4(), HttpCreateTcp4TxEvent(), HttpDns4(), HttpGenRequestString(), HTTP_TOKEN_WRAP::HttpInstance, HttpTcpNotReady(), HTTP_TOKEN_WRAP::HttpToken, HttpTokenExist(), HttpTransmitTcp4(), _HTTP_PROTOCOL::LocalAddressIsIPv6, HTTP_TCP_TOKEN_WRAP::Method, _HTTP_PROTOCOL::RemoteAddr, _HTTP_PROTOCOL::RemoteHost, _HTTP_PROTOCOL::RemotePort, _HTTP_PROTOCOL::State, _HTTP_PROTOCOL::Tcp4, HTTP_TOKEN_WRAP::TcpWrap, HTTP_TCP_TOKEN_WRAP::TxToken, _HTTP_PROTOCOL::TxTokens, and _HTTP_PROTOCOL::Url.

```
EFI_STATUS EFIAPI EfiHttpResponse ( IN EFI_HTTP_PROTOCOL *  This,
                                    IN EFI_HTTP_TOKEN *     Token
                                  )
```

The Response() function queues an HTTP response to this HTTP instance, similar to Receive() function in the EFI TCP driver. When the HTTP request is sent successfully, or if there is an error, Status in token will be updated and Event will be signaled.

The HTTP driver will queue a receive token to the underlying TCP instance. When data is received in the underlying TCP instance, the data will be parsed and Token will be populated with the response data. If the data received from the remote host contains an incomplete or invalid HTTP header, the HTTP driver will continue waiting (asynchronously) for more data to be sent from the remote host before signaling Event in Token.

It is the responsibility of the caller to allocate a buffer for Body and specify the size in BodyLength. If the remote host provides a response that contains a content body, up to BodyLength bytes will be copied from the receive buffer into Body and BodyLength will be updated with the amount of bytes received and copied to Body. This allows the client to download a large file in chunks instead of into one contiguous block of memory. Similar to HTTP request, if Body is not NULL and BodyLength is non-zero and all other fields are NULL or 0, the HTTP driver will queue a receive token to underlying TCP instance. If data arrives in the receive buffer, up to BodyLength bytes of data will be copied to Body. The HTTP driver will then update BodyLength with the amount of bytes received and copied to Body.

If the HTTP driver does not have an open underlying TCP connection with the host specified in the response URL, Request() will return EFI_ACCESS_DENIED. This is consistent with RFC 2616 recommendation that HTTP clients should attempt to maintain an open TCP connection between client and host.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_PROTOCOL instance.
[in]	*Token*	Pointer to storage containing HTTP response token.

Return values:

EFI_SUCCESS	Allocation succeeded.
EFI_NOT_STARTED	This EFI HTTP Protocol instance has not been initialized.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. Token is NULL. Token->Message->Headers is NULL. Token->Message is NULL. Token->Message->Body is not NULL, Token->Message->BodyLength is non-zero, and Token->Message->Data is NULL, but a previous call to Response() has not been completed successfully.
EFI_OUT_OF_RESOURCES	Could not allocate enough system resources.
EFI_ACCESS_DENIED	An open TCP connection is not present with the host specified by response URL.

References HTTP_INSTANCE_FROM_PROTOCOL, HTTP_STATE_TCP_CONNECTED, HttpCreateTcp4RxEvent(), HTTP_TOKEN_WRAP::HttpInstance, HttpResponseWorker(), HTTP_TOKEN_WRAP::HttpToken, HttpTokenExist(), _HTTP_PROTOCOL::LocalAddressIsIPv6, HTTP_TCP_TOKEN_WRAP::RxToken, _HTTP_PROTOCOL::RxTokens, _HTTP_PROTOCOL::State, and HTTP_TOKEN_WRAP::TcpWrap.

Variable Documentation

EFI_HTTP_PROTOCOL mEfiHttpTemplate

Referenced by HttpInitProtocol().

NetworkPkg/HttpDxe/HttpProto.c File Reference

Functions

VOID EFIAPI HttpCommonNotify (IN EFI_EVENT Event, IN VOID *Context)
VOID EFIAPI HttpTcpTransmitNotifyDpc (IN VOID *Context)
VOID EFIAPI HttpTcpTransmitNotify (IN EFI_EVENT Event, IN VOID *Context)
VOID EFIAPI HttpTcpReceiveNotifyDpc (IN VOID *Context)
VOID EFIAPI HttpTcpReceiveNotify (IN EFI_EVENT Event, IN VOID *Context)
EFI_STATUS HttpCreateTcp4ConnCloseEvent (IN HTTP_PROTOCOL *HttpInstance)
VOID HttpCloseTcp4ConnCloseEvent (IN HTTP_PROTOCOL *HttpInstance)
EFI_STATUS HttpCreateTcp4TxEvent (IN HTTP_TOKEN_WRAP *Wrap)
EFI_STATUS HttpCreateTcp4RxEventForHeader (IN HTTP_PROTOCOL *HttpInstance)
EFI_STATUS HttpCreateTcp4RxEvent (IN HTTP_TOKEN_WRAP *Wrap)
EFI_STATUS HttpInitProtocol (IN HTTP_SERVICE *HttpSb, IN OUT HTTP_PROTOCOL *HttpInstance)
VOID HttpCleanProtocol (IN HTTP_PROTOCOL *HttpInstance)
EFI_STATUS HttpCreateConnection (IN HTTP_PROTOCOL *HttpInstance)
EFI_STATUS HttpCloseConnection (IN HTTP_PROTOCOL *HttpInstance)
EFI_STATUS HttpConfigureTcp4 (IN HTTP_PROTOCOL *HttpInstance, IN HTTP_TOKEN_WRAP *Wrap)
EFI_STATUS HttpConnectTcp4 (IN HTTP_PROTOCOL *HttpInstance)
EFI_STATUS HttpTransmitTcp4 (IN HTTP_PROTOCOL *HttpInstance, IN HTTP_TOKEN_WRAP *Wrap, IN UINT8 *TxString, IN UINTN TxStringLen)
EFI_HTTP_STATUS_CODE HttpMappingToStatusCode (IN UINTN StatusCode)
EFI_STATUS EFIAPI HttpTokenExist (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Context)
EFI_STATUS EFIAPI HttpTcpNotReady (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Context)
EFI_STATUS EFIAPI HttpTcpTransmit (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Context)
EFI_STATUS EFIAPI HttpTcpReceive (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Context)
CHAR8 * HttpGenRequestString (IN HTTP_PROTOCOL *HttpInstance, IN EFI_HTTP_MESSAGE *Message, IN CHAR8 *Url)

Detailed Description

Miscellaneous routines for HttpDxe driver.

Copyright (c) 2015, Intel Corporation. All rights reserved.
(C) Copyright 2015 Hewlett Packard Enterprise Development LP
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

VOID HttpCleanProtocol (IN HTTP_PROTOCOL * *HttpInstance*)

Clean up the HTTP child, release all the resources used by it.

Parameters:

[in] *HttpInstance* The HTTP child to clean up.

References HttpCloseConnection(), and HttpCloseTcp4ConnCloseEvent().

Referenced by EfiHttpConfigure(), HttpServiceBindingCreateChild(), and HttpServiceBindingDestroyChild().

EFI_STATUS HttpCloseConnection (IN HTTP_PROTOCOL * *HttpInstance*)

Close existing TCP connection.

Parameters:

[in] *HttpInstance* The HTTP instance private data.

Return values:

EFI_SUCCESS The TCP connection is closed.

Others Other error as indicated.

References HTTP_STATE_TCP_CLOSED, and HTTP_STATE_TCP_CONNECTED.

Referenced by EfiHttpRequest(), HttpCleanProtocol(), and HttpConnectTcp4().

VOID HttpCloseTcp4ConnCloseEvent (IN HTTP_PROTOCOL * *HttpInstance*)

Close events in the TCP4 connection token and TCP4 close token.

Parameters:

[in] *HttpInstance* Pointer to HTTP_PROTOCOL structure.

Referenced by EfiHttpRequest(), HttpCleanProtocol(), and HttpCreateTcp4ConnCloseEvent().

VOID EFIAPI HttpCommonNotify (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

The common notify function used in HTTP driver.

Parameters:

[in] *Event* The event signaled.

[in] *Context* The context.

Referenced by HttpCreateTcp4ConnCloseEvent(), HttpCreateTcp4RxEventForHeader(), and HttpDns4().

EFI_STATUS HttpConfigureTcp4 (IN HTTP_PROTOCOL * *HttpInstance*,
IN HTTP_TOKEN_WRAP * *Wrap*

)

Configure TCP4 protocol child.

Parameters:

[in]	*HttpInstance*	The HTTP instance private data.
[in]	*Wrap*	The HTTP token's wrap data.

Return values:

EFI_SUCCESS	The TCP4 protocol child is configured.
Others	Other error as indicated.

References HTTP_BUFFER_SIZE_DEAULT, HTTP_CONNECTION_TIMEOUT, HTTP_DATA_RETRIES, HTTP_FIN_TIMEOUT, HTTP_KEEP_ALIVE_INTERVAL, HTTP_KEEP_ALIVE_PROBES, HTTP_KEEP_ALIVE_TIME, HTTP_MAX_SYN_BACK_LOG, HTTP_STATE_TCP_CONFIGED, HTTP_TOS_DEAULT, HTTP_TTL_DEAULT, HttpCreateTcp4ConnCloseEvent(), and HttpCreateTcp4TxEvent().

Referenced by EfiHttpRequest().

EFI_STATUS HttpConnectTcp4 (IN HTTP_PROTOCOL * *HttpInstance*)

Check existing TCP connection, if in error state, receover TCP4 connection.

Parameters:

[in]	*HttpInstance*	The HTTP instance private data.

Return values:

EFI_SUCCESS	The TCP connection is established.
EFI_NOT_READY	TCP4 protocol child is not created or configured.
Others	Other error as indicated.

References HTTP_STATE_TCP_CONFIGED, HttpCloseConnection(), and HttpCreateConnection().

Referenced by EfiHttpRequest().

EFI_STATUS HttpCreateConnection (IN HTTP_PROTOCOL * *HttpInstance*)

Establish TCP connection with HTTP server.

Parameters:

[in]	*HttpInstance*	The HTTP instance private data.

Return values:

EFI_SUCCESS	The TCP connection is established.
Others	Other error as indicated.

References HTTP_STATE_TCP_CONNECTED.

Referenced by HttpConnectTcp4().

EFI_STATUS HttpCreateTcp4ConnCloseEvent (IN HTTP_PROTOCOL * *HttpInstance*)

Create events for the TCP4 connection token and TCP4 close token.

Parameters:

[in] *HttpInstance* Pointer to HTTP_PROTOCOL structure.

Return values:

EFI_SUCCESS The events are created successfully.
others Other error as indicated.

References HttpCloseTcp4ConnCloseEvent(), and HttpCommonNotify().

Referenced by HttpConfigureTcp4().

EFI_STATUS HttpCreateTcp4RxEvent (IN HTTP_TOKEN_WRAP * *Wrap*)
Create event for the TCP4 receive token which is used to receive HTTP body.

Parameters:

[in] *Wrap* Point to HTTP token's wrap data.

Return values:

EFI_SUCCESS The events is created successfully.
others Other error as indicated.

References HttpTcpReceiveNotify(), HTTP_TCP_TOKEN_WRAP::RxData, and HTTP_TCP_TOKEN_WRAP::RxToken.

Referenced by EfiHttpResponse().

EFI_STATUS HttpCreateTcp4RxEventForHeader (IN HTTP_PROTOCOL * *HttpInstance*)
Create event for the TCP4 receive token which is used to receive HTTP header.

Parameters:

[in] *HttpInstance* Pointer to HTTP_PROTOCOL structure.

Return values:

EFI_SUCCESS The events is created successfully.
others Other error as indicated.

References HttpCommonNotify().

Referenced by HttpResponseWorker().

EFI_STATUS HttpCreateTcp4TxEvent (IN HTTP_TOKEN_WRAP * *Wrap*)
Create event for the TCP4 transmit token.

Parameters:

[in] *Wrap* Point to HTTP token's wrap data.

Return values:

EFI_SUCCESS The events is created successfully.
others Other error as indicated.

References HttpTcpTransmitNotify(), HTTP_TCP_TOKEN_WRAP::TxData, and HTTP_TCP_TOKEN_WRAP::TxToken.

Referenced by EfiHttpRequest(), and HttpConfigureTcp4().

```
CHAR8* HttpGenRequestString ( IN HTTP_PROTOCOL *        HttpInstance,
                              IN EFI_HTTP_MESSAGE *     Message,
                              IN CHAR8 *                Url
                            )
```

Generate HTTP request string.

Parameters:

[in]	*HttpInstance*	Pointer to HTTP_PROTOCOL structure.
[in]	*Message*	Pointer to storage containing HTTP message data.
[in]	*Url*	The URL of a remote host.

Returns:

Pointer to the created HTTP request string.

NULL if any error occured.

References HTTP_GET_STR, HTTP_HEAD_STR, HTTP_MAXIMUM_METHOD_LEN, HTTP_VERSION_CRLF_STR, and mHttpUtilities.

Referenced by EfiHttpRequest(), and HttpTcpTransmit().

```
EFI_STATUS HttpInitProtocol ( IN HTTP_SERVICE *          HttpSb,
                              IN OUT HTTP_PROTOCOL *     HttpInstance
                            )
```

Intiialize the HTTP_PROTOCOL structure to the unconfigured state.

Parameters:

[in]	*HttpSb*	The HTTP service private instance.
[in,out]	*HttpInstance*	Pointer to HTTP_PROTOCOL structure.

Return values:

EFI_SUCCESS	HTTP_PROTOCOL structure is initialized successfully.
Others	Other error as indicated.

References HTTP_PROTOCOL_SIGNATURE, HTTP_URL_BUFFER_LEN, and mEfiHttpTemplate.

Referenced by EfiHttpConfigure(), and HttpServiceBindingCreateChild().

```
EFI_HTTP_STATUS_CODE HttpMappingToStatusCode ( IN UINTN  StatusCode  )
```

Translate the status code in HTTP message to EFI_HTTP_STATUS_CODE defined in UEFI 2.5 specification.

Parameters:

[in]	*StatusCode*	The status code value in HTTP message.

Returns:

Value defined in EFI_HTTP_STATUS_CODE .

Referenced by HttpResponseWorker().

```
EFI_STATUS EFIAPI HttpTcpNotReady ( IN NET_MAP *        Map,
                                    IN NET_MAP_ITEM *   Item,
                                    IN VOID *           Context
                                  )
```

Check whether the HTTP message associated with TxToken is already sent out.

Parameters:

[in]	*Map*	The container of TxToken.
[in]	*Item*	Current item to check against.
[in]	*Context*	The Token to check againist.

Return values:

EFI_NOT_READY	The HTTP message is still queued in the list.
EFI_SUCCESS	The HTTP message has been sent out.

References HTTP_TCP_TOKEN_WRAP::IsTxDone, and HTTP_TOKEN_WRAP::TcpWrap.

Referenced by EfiHttpRequest().

```
EFI_STATUS EFIAPI HttpTcpReceive ( IN NET_MAP *        Map,
                                   IN NET_MAP_ITEM *   Item,
                                   IN VOID *           Context
                                 )
```

Receive the HTTP response by processing the associated HTTP token.

Parameters:

[in]	*Map*	The container of RxToken.
[in]	*Item*	Current item to check against.
[in]	*Context*	The Token to check againist.

Return values:

EFI_SUCCESS	The HTTP response is queued into TCP receive queue.
Others	Other error as indicated.

References HttpResponseWorker().

Referenced by HttpTcpReceiveNotifyDpc().

```
VOID EFIAPI HttpTcpReceiveNotify ( IN EFI_EVENT   Event,
                                   IN VOID *      Context
                                 )
```

Request HttpTcpReceiveNotifyDpc as a DPC at TPL_CALLBACK.

Parameters:

Event The receive event delivered to TCP for receive.
Context Context for the callback.

References HttpTcpReceiveNotifyDpc().

Referenced by HttpCreateTcp4RxEvent().

VOID EFIAPI HttpTcpReceiveNotifyDpc (IN VOID * *Context*)

The notify function associated with RxToken for Tcp4->Receive ().

Parameters:

[in] *Context* The context.

References _HTTP_PROTOCOL::CacheBody, _HTTP_PROTOCOL::CacheLen, _HTTP_PROTOCOL::CacheOffset, HTTP_TOKEN_WRAP::HttpInstance, HttpTcpReceive(), HTTP_TOKEN_WRAP::HttpToken, HTTP_TCP_TOKEN_WRAP::IsRxDone, _HTTP_PROTOCOL::MsgParser, _HTTP_PROTOCOL::NextMsg, HTTP_TCP_TOKEN_WRAP::RxData, HTTP_TCP_TOKEN_WRAP::RxToken, _HTTP_PROTOCOL::RxTokens, and HTTP_TOKEN_WRAP::TcpWrap.

Referenced by HttpTcpReceiveNotify().

EFI_STATUS EFIAPI HttpTcpTransmit (IN NET_MAP * *Map*,
IN NET_MAP_ITEM * *Item*,
IN VOID * *Context*
)

Transmit the HTTP mssage by processing the associated HTTP token.

Parameters:

[in] *Map* The container of TxToken.
[in] *Item* Current item to check against.
[in] *Context* The Token to check againist.

Return values:

EFI_OUT_OF_RESOURCES Failed to allocate resources.
EFI_SUCCESS The HTTP message is queued into TCP transmit queue.

References HttpGenRequestString(), HTTP_TOKEN_WRAP::HttpInstance, HTTP_TOKEN_WRAP::HttpToken, HttpTransmitTcp4(), HTTP_TCP_TOKEN_WRAP::IsTxDone, and HTTP_TOKEN_WRAP::TcpWrap.

Referenced by HttpTcpTransmitNotifyDpc().

VOID EFIAPI HttpTcpTransmitNotify (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

Request HttpTcpTransmitNotifyDpc as a DPC at TPL_CALLBACK.

Parameters:

Event The receive event delivered to TCP for transmit.
Context Context for the callback.

References HttpTcpTransmitNotifyDpc().

Referenced by HttpCreateTcp4TxEvent().

VOID EFIAPI HttpTcpTransmitNotifyDpc (IN VOID * *Context*)

The notify function associated with TxToken for Tcp4->Transmit().

Parameters:

[in] *Context* The context.

References HTTP_TOKEN_WRAP::HttpInstance, HttpTcpTransmit(), HTTP_TOKEN_WRAP::HttpToken, HTTP_TCP_TOKEN_WRAP::IsTxDone, HTTP_TOKEN_WRAP::TcpWrap, HTTP_TCP_TOKEN_WRAP::TxToken, and _HTTP_PROTOCOL::TxTokens.

Referenced by HttpTcpTransmitNotify().

```
EFI_STATUS EFIAPI HttpTokenExist ( IN NET_MAP *          Map,
                                   IN NET_MAP_ITEM *     Item,
                                   IN VOID *             Context
                                 )
```

Check whether the user's token or event has already been enqueue on HTTP TxToken or RxToken list.

Parameters:

[in] *Map* The container of either user's transmit or receive token.
[in] *Item* Current item to check against.
[in] *Context* The Token to check againist.

Return values:

EFI_ACCESS_DENIED The token or event has already been enqueued in IP
EFI_SUCCESS The current item isn't the same token/event as the context.

Referenced by EfiHttpRequest(), and EfiHttpResponse().

```
EFI_STATUS HttpTransmitTcp4 ( IN HTTP_PROTOCOL *      HttpInstance,
                              IN HTTP_TOKEN_WRAP *    Wrap,
                              IN UINT8 *              TxString,
                              IN UINTN                TxStringLen
                            )
```

Send the HTTP message through TCP4.

Parameters:

[in]	*HttpInstance*	The HTTP instance private data.
[in]	*Wrap*	The HTTP token's wrap data.
[in]	*TxString*	Buffer containing the HTTP message string.
[in]	*TxStringLen*	Length of the HTTP message string in bytes.

Return values:

EFI_SUCCESS	The HTTP message is queued into TCP transmit queue.
Others	Other error as indicated.

Referenced by EfiHttpRequest(), and HttpTcpTransmit().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/HttpDxe/HttpProto.h File Reference

Data Structures

struct _HTTP_SERVICE
struct HTTP_TCP_TOKEN_WRAP
struct _HTTP_PROTOCOL
struct HTTP_TOKEN_WRAP

Defines

"memItemRight" valign="bottom">DEF_BUF_LEN 2048
#define HTTP_SERVICE_SIGNATURE SIGNATURE_32('H', 't', 't', 'S')
#define HTTP_SERVICE_FROM_PROTOCOL(a)
#define HTTP_STATE_UNCONFIGED 0
#define HTTP_STATE_HTTP_CONFIGED 1
#define HTTP_STATE_TCP_CONFIGED 2
#define HTTP_STATE_TCP_UNCONFIGED 3
#define HTTP_STATE_TCP_CONNECTED 4
#define HTTP_STATE_TCP_CLOSED 5
#define HTTP_TOS_DEAULT 8
#define HTTP_TTL_DEAULT 255
#define HTTP_BUFFER_SIZE_DEAULT 65535
#define HTTP_MAX_SYN_BACK_LOG 5
#define HTTP_CONNECTION_TIMEOUT 60
#define HTTP_DATA_RETRIES 12
#define HTTP_FIN_TIMEOUT 2
#define HTTP_KEEP_ALIVE_PROBES 6
#define HTTP_KEEP_ALIVE_TIME 7200
#define HTTP_KEEP_ALIVE_INTERVAL 30
#define HTTP_URL_BUFFER_LEN 4096
#define HTTP_PROTOCOL_SIGNATURE SIGNATURE_32('H' 't', 't', 'P')
#define HTTP_INSTANCE_FROM_PROTOCOL(a)

Typedefs

typedef struct _HTTP_SERVICE HTTP_SERVICE
typedef struct _HTTP_PROTOCOL HTTP_PROTOCOL

Functions

VOID EFIAPI HttpCommonNotify (IN EFI_EVENT Event, IN VOID *Context)
EFI_STATUS HttpCreateTcp4ConnCloseEvent (IN HTTP_PROTOCOL *HttpInstance)
VOID HttpCloseTcp4ConnCloseEvent (IN HTTP_PROTOCOL *HttpInstance)
EFI_STATUS HttpCreateTcp4TxEvent (IN

HTTP_TOKEN_WRAP *Wrap)
EFI_STATUS HttpCreateTcp4RxEventForHeader (IN HTTP_PROTOCOL *HttpInstance)
EFI_STATUS HttpCreateTcp4RxEvent (IN HTTP_TOKEN_WRAP *Wrap)
EFI_STATUS HttpInitProtocol (IN HTTP_SERVICE *HttpSb, IN OUT HTTP_PROTOCOL *HttpInstance)
VOID HttpCleanProtocol (IN HTTP_PROTOCOL *HttpInstance)
EFI_STATUS HttpCreateConnection (IN HTTP_PROTOCOL *HttpInstance)
EFI_STATUS HttpCloseConnection (IN HTTP_PROTOCOL *HttpInstance)
EFI_STATUS HttpConfigureTcp4 (IN HTTP_PROTOCOL *HttpInstance, IN HTTP_TOKEN_WRAP *Wrap)
EFI_STATUS HttpConnectTcp4 (IN HTTP_PROTOCOL *HttpInstance)
EFI_STATUS HttpTransmitTcp4 (IN HTTP_PROTOCOL *HttpInstance, IN HTTP_TOKEN_WRAP *Wrap, IN UINT8 *TxString, IN UINTN TxStringLen)
EFI_HTTP_STATUS_CODE HttpMappingToStatusCode (IN UINTN StatusCode)
EFI_STATUS EFIAPI HttpTokenExist (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Context)
EFI_STATUS EFIAPI HttpTcpNotReady (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Context)
EFI_STATUS EFIAPI HttpTcpTransmit (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Context)
EFI_STATUS EFIAPI HttpTcpReceive (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Context)
CHAR8 * HttpGenRequestString (IN HTTP_PROTOCOL *HttpInstance, IN EFI_HTTP_MESSAGE *Message, IN CHAR8 *Url)
EFI_STATUS HttpResponseWorker (IN HTTP_TOKEN_WRAP *Wrap)

Detailed Description

The header files of miscellaneous routines for HttpDxe driver.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by HttpResponseWorker().

#define HTTP_BUFFER_SIZE_DEAULT 65535

Referenced by HttpConfigureTcp4().

#define HTTP_CONNECTION_TIMEOUT 60

Referenced by HttpConfigureTcp4().

#define HTTP_DATA_RETRIES 12

Referenced by HttpConfigureTcp4().

#define HTTP_FIN_TIMEOUT 2

Referenced by HttpConfigureTcp4().

#define HTTP_INSTANCE_FROM_PROTOCOL (a)

Value:

```
CR ( \
  (a), \
  HTTP_PROTOCOL, \
  Http, \
  HTTP_PROTOCOL_SIGNATURE \
  )
```

Referenced by EfiHttpCancel(), EfiHttpConfigure(), EfiHttpGetModeData(), EfiHttpPoll(), EfiHttpRequest(), EfiHttpResponse(), and HttpServiceBindingDestroyChild().

#define HTTP_KEEP_ALIVE_INTERVAL 30

Referenced by HttpConfigureTcp4().

#define HTTP_KEEP_ALIVE_PROBES 6

Referenced by HttpConfigureTcp4().

#define HTTP_KEEP_ALIVE_TIME 7200

Referenced by HttpConfigureTcp4().

#define HTTP_MAX_SYN_BACK_LOG 5

Referenced by HttpConfigureTcp4().

#define HTTP_PROTOCOL_SIGNATURE SIGNATURE_32('H', 't', 't', 'P')

Referenced by HttpDestroyChildEntryInHandleBuffer(), and HttpInitProtocol().

#define HTTP_SERVICE_FROM_PROTOCOL (a)

Value:

```
CR ( \
  (a), \
  HTTP_SERVICE, \
  ServiceBinding, \
  HTTP_SERVICE_SIGNATURE \
  )
```

Referenced by HttpDxeDriverBindingStop(), HttpServiceBindingCreateChild(), and HttpServiceBindingDestroyChild().

#define HTTP_SERVICE_SIGNATURE SIGNATURE_32('H', 't', 't', 'S')

Referenced by HttpCreateService().

#define HTTP_STATE_HTTP_CONFIGED 1

Referenced by EfiHttpConfigure(), EfiHttpGetModeData(), and EfiHttpRequest().

#define HTTP_STATE_TCP_CLOSED 5

Referenced by HttpCloseConnection().

#define HTTP_STATE_TCP_CONFIGED 2

Referenced by HttpConfigureTcp4(), and HttpConnectTcp4().

#define HTTP_STATE_TCP_CONNECTED 4

Referenced by EfiHttpCancel(), EfiHttpPoll(), EfiHttpResponse(), HttpCloseConnection(), and HttpCreateConnection().

#define HTTP_STATE_TCP_UNCONFIGED 3

#define HTTP_STATE_UNCONFIGED 0

Referenced by EfiHttpConfigure().

#define HTTP_TOS_DEAULT 8

Referenced by HttpConfigureTcp4().

#define HTTP_TTL_DEAULT 255

Referenced by HttpConfigureTcp4().

#define HTTP_URL_BUFFER_LEN 4096

Referenced by EfiHttpRequest(), and HttpInitProtocol().

Typedef Documentation

typedef struct _HTTP_PROTOCOL HTTP_PROTOCOL

typedef struct _HTTP_SERVICE HTTP_SERVICE

Function Documentation

VOID HttpCleanProtocol (IN HTTP_PROTOCOL * *HttpInstance*)
Clean up the HTTP child, release all the resources used by it.

Parameters:

`[in]` *HttpInstance* The HTTP child to clean up.

References HttpCloseConnection(), and HttpCloseTcp4ConnCloseEvent().

Referenced by EfiHttpConfigure(), HttpServiceBindingCreateChild(), and HttpServiceBindingDestroyChild().

EFI_STATUS HttpCloseConnection (IN HTTP_PROTOCOL * *HttpInstance*)
Close existing TCP connection.

Parameters:

`[in]` *HttpInstance* The HTTP instance private data.

Return values:

EFI_SUCCESS The TCP connection is closed.
Others Other error as indicated.

References HTTP_STATE_TCP_CLOSED, and HTTP_STATE_TCP_CONNECTED.

Referenced by EfiHttpRequest(), HttpCleanProtocol(), and HttpConnectTcp4().

VOID HttpCloseTcp4ConnCloseEvent (IN HTTP_PROTOCOL * *HttpInstance*)

Close events in the TCP4 connection token and TCP4 close token.

Parameters:

[in] *HttpInstance* Pointer to HTTP_PROTOCOL structure.

Referenced by EfiHttpRequest(), HttpCleanProtocol(), and HttpCreateTcp4ConnCloseEvent().

VOID EFIAPI HttpCommonNotify (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

The common notify function used in HTTP driver.

Parameters:

[in] *Event* The event signaled.
[in] *Context* The context.

Referenced by HttpCreateTcp4ConnCloseEvent(), HttpCreateTcp4RxEventForHeader(), and HttpDns4().

EFI_STATUS HttpConfigureTcp4 (IN HTTP_PROTOCOL * *HttpInstance*,
IN HTTP_TOKEN_WRAP * *Wrap*
)

Configure TCP4 protocol child.

Parameters:

[in] *HttpInstance* The HTTP instance private data.
[in] *Wrap* The HTTP token's wrap data.

Return values:

EFI_SUCCESS The TCP4 protocol child is configured.
Others Other error as indicated.

References HTTP_BUFFER_SIZE_DEAULT, HTTP_CONNECTION_TIMEOUT, HTTP_DATA_RETRIES, HTTP_FIN_TIMEOUT, HTTP_KEEP_ALIVE_INTERVAL, HTTP_KEEP_ALIVE_PROBES, HTTP_KEEP_ALIVE_TIME, HTTP_MAX_SYN_BACK_LOG, HTTP_STATE_TCP_CONFIGED, HTTP_TOS_DEAULT, HTTP_TTL_DEAULT, HttpCreateTcp4ConnCloseEvent(), and HttpCreateTcp4TxEvent().

Referenced by EfiHttpRequest().

EFI_STATUS HttpConnectTcp4 (IN HTTP_PROTOCOL * *HttpInstance*)

Check existing TCP connection, if in error state, receover TCP4 connection.

Parameters:

[in] *HttpInstance* The HTTP instance private data.

Return values:

EFI_SUCCESS	The TCP connection is established.
EFI_NOT_READY	TCP4 protocol child is not created or configured.
Others	Other error as indicated.

References HTTP_STATE_TCP_CONFIGED, HttpCloseConnection(), and HttpCreateConnection().

Referenced by EfiHttpRequest().

EFI_STATUS HttpCreateConnection (IN HTTP_PROTOCOL * *HttpInstance*)
Establish TCP connection with HTTP server.

Parameters:

`[in]` *HttpInstance* The HTTP instance private data.

Return values:

EFI_SUCCESS	The TCP connection is established.
Others	Other error as indicated.

References HTTP_STATE_TCP_CONNECTED.

Referenced by HttpConnectTcp4().

EFI_STATUS HttpCreateTcp4ConnCloseEvent (IN HTTP_PROTOCOL * *HttpInstance*)
Create events for the TCP4 connection token and TCP4 close token.

Parameters:

`[in]` *HttpInstance* Pointer to HTTP_PROTOCOL structure.

Return values:

EFI_SUCCESS	The events are created successfully.
others	Other error as indicated.

References HttpCloseTcp4ConnCloseEvent(), and HttpCommonNotify().

Referenced by HttpConfigureTcp4().

EFI_STATUS HttpCreateTcp4RxEvent (IN HTTP_TOKEN_WRAP * *Wrap*)
Create event for the TCP4 receive token which is used to receive HTTP body.

Parameters:

`[in]` *Wrap* Point to HTTP token's wrap data.

Return values:

EFI_SUCCESS	The events is created successfully.
others	Other error as indicated.

References HttpTcpReceiveNotify(), HTTP_TCP_TOKEN_WRAP::RxData, and HTTP_TCP_TOKEN_WRAP::RxToken.

Referenced by EfiHttpResponse().

EFI_STATUS HttpCreateTcp4RxEventForHeader (IN HTTP_PROTOCOL * *HttpInstance*)

Create event for the TCP4 receive token which is used to receive HTTP header.

Parameters:

[in] *HttpInstance* Pointer to HTTP_PROTOCOL structure.

Return values:

EFI_SUCCESS The events is created successfully.
others Other error as indicated.

References HttpCommonNotify().

Referenced by HttpResponseWorker().

EFI_STATUS HttpCreateTcp4TxEvent (IN HTTP_TOKEN_WRAP * *Wrap*)

Create event for the TCP4 transmit token.

Parameters:

[in] *Wrap* Point to HTTP token's wrap data.

Return values:

EFI_SUCCESS The events is created successfully.
others Other error as indicated.

References HttpTcpTransmitNotify(), HTTP_TCP_TOKEN_WRAP::TxData, and HTTP_TCP_TOKEN_WRAP::TxToken.

Referenced by EfiHttpRequest(), and HttpConfigureTcp4().

CHAR8* HttpGenRequestString (IN HTTP_PROTOCOL * *HttpInstance*,
IN EFI_HTTP_MESSAGE * *Message*,
IN CHAR8 * *Url*
)

Generate HTTP request string.

Parameters:

[in] *HttpInstance* Pointer to HTTP_PROTOCOL structure.
[in] *Message* Pointer to storage containing HTTP message data.
[in] *Url* The URL of a remote host.

Returns:

Pointer to the created HTTP request string.

NULL if any error occured.

References HTTP_GET_STR, HTTP_HEAD_STR, HTTP_MAXIMUM_METHOD_LEN, HTTP_VERSION_CRLF_STR, and mHttpUtilities.

Referenced by EfiHttpRequest(), and HttpTcpTransmit().

EFI_STATUS HttpInitProtocol (IN HTTP_SERVICE * *HttpSb*,
IN OUT HTTP_PROTOCOL * *HttpInstance*
)

Intiialize the HTTP_PROTOCOL structure to the unconfigured state.

Parameters:

[in]	*HttpSb*	The HTTP service private instance.
[in,out]	*HttpInstance*	Pointer to HTTP_PROTOCOL structure.

Return values:

EFI_SUCCESS	HTTP_PROTOCOL structure is initialized successfully.
Others	Other error as indicated.

References HTTP_PROTOCOL_SIGNATURE, HTTP_URL_BUFFER_LEN, and mEfiHttpTemplate.

Referenced by EfiHttpConfigure(), and HttpServiceBindingCreateChild().

EFI_HTTP_STATUS_CODE HttpMappingToStatusCode (IN UINTN *StatusCode*)

Translate the status code in HTTP message to EFI_HTTP_STATUS_CODE defined in UEFI 2.5 specification.

Parameters:

[in]	*StatusCode*	The status code value in HTTP message.

Returns:

Value defined in EFI_HTTP_STATUS_CODE .

Referenced by HttpResponseWorker().

EFI_STATUS HttpResponseWorker (IN HTTP_TOKEN_WRAP * *Wrap*)

The work function of EfiHttpResponse().

Parameters:

[in]	*Wrap*	Pointer to HTTP token's wrap data.

Return values:

EFI_SUCCESS	Allocation succeeded.
EFI_OUT_OF_RESOURCES	Failed to complete the opration due to lack of resources.
EFI_NOT_READY	Can't find a corresponding TxToken.

The work function of EfiHttpResponse().

Parameters:

[in]	*Wrap*	Pointer to HTTP token's wrap data.

Return values:

EFI_SUCCESS	Allocation succeeded.
EFI_OUT_OF_RESOURCES	Failed to complete the opration due to lack of resources.

EFI_NOT_READY	Can't find a corresponding TxToken or the EFI_HTTP_UTILITIES_PROTOCOL is not available.

References _HTTP_PROTOCOL::CacheBody, _HTTP_PROTOCOL::CacheLen, _HTTP_PROTOCOL::CacheOffset, DEF_BUF_LEN, HTTP_CRLF_STR, HTTP_END_OF_HDR_STR, HTTP_VERSION_STR, HttpBodyParserCallback(), HttpCreateTcp4RxEventForHeader(), HttpMappingToStatusCode(), HTTP_TOKEN_WRAP::HttpToken, _HTTP_PROTOCOL::IsRxDone, mHttpUtilities, _HTTP_PROTOCOL::MsgParser, _HTTP_PROTOCOL::NextMsg, _HTTP_PROTOCOL::RxToken, _HTTP_PROTOCOL::Tcp4, and _HTTP_PROTOCOL::TxTokens.

Referenced by EfiHttpResponse(), and HttpTcpReceive().

```
EFI_STATUS EFIAPI HttpTcpNotReady ( IN NET_MAP *        Map,
                                    IN NET_MAP_ITEM *   Item,
                                    IN VOID *           Context
                                  )
```

Check whether the HTTP message associated with TxToken is already sent out.

Parameters:

[in]	*Map*	The container of TxToken.
[in]	*Item*	Current item to check against.
[in]	*Context*	The Token to check againist.

Return values:

EFI_NOT_READY	The HTTP message is still queued in the list.
EFI_SUCCESS	The HTTP message has been sent out.

References HTTP_TCP_TOKEN_WRAP::IsTxDone, and HTTP_TOKEN_WRAP::TcpWrap.

Referenced by EfiHttpRequest().

```
EFI_STATUS EFIAPI HttpTcpReceive ( IN NET_MAP *        Map,
                                   IN NET_MAP_ITEM *   Item,
                                   IN VOID *           Context
                                 )
```

Receive the HTTP response by processing the associated HTTP token.

Parameters:

[in]	*Map*	The container of RxToken.
[in]	*Item*	Current item to check against.
[in]	*Context*	The Token to check againist.

Return values:

EFI_SUCCESS	The HTTP response is queued into TCP receive queue.
Others	Other error as indicated.

References HttpResponseWorker().

Referenced by HttpTcpReceiveNotifyDpc().

```
EFI_STATUS EFIAPI HttpTcpTransmit ( IN NET_MAP *          Map,
                                    IN NET_MAP_ITEM *     Item,
                                    IN VOID *             Context
                                  )
```

Transmit the HTTP mssage by processing the associated HTTP token.

Parameters:

[in]	*Map*	The container of TxToken.
[in]	*Item*	Current item to check against.
[in]	*Context*	The Token to check againist.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate resources.
EFI_SUCCESS	The HTTP message is queued into TCP transmit queue.

References HttpGenRequestString(), HTTP_TOKEN_WRAP::HttpInstance, HTTP_TOKEN_WRAP::HttpToken, HttpTransmitTcp4(), HTTP_TCP_TOKEN_WRAP::IsTxDone, and HTTP_TOKEN_WRAP::TcpWrap.

Referenced by HttpTcpTransmitNotifyDpc().

```
EFI_STATUS EFIAPI HttpTokenExist ( IN NET_MAP *       Map,
                                   IN
                                   NET_MAP_ITEM       Item,
                                   *
                                   IN VOID *          Context
                                 )
```

Check whether the user's token or event has already been enqueue on HTTP TxToken or RxToken list.

Parameters:

[in]	*Map*	The container of either user's transmit or receive token.
[in]	*Item*	Current item to check against.
[in]	*Context*	The Token to check againist.

Return values:

EFI_ACCESS_DENIED	The token or event has already been enqueued in IP
EFI_SUCCESS	The current item isn't the same token/event as the context.

Referenced by EfiHttpRequest(), and EfiHttpResponse().

```
EFI_STATUS HttpTransmitTcp4 ( IN HTTP_PROTOCOL *      HttpInstance,
                              IN HTTP_TOKEN_WRAP *    Wrap,
                              IN UINT8 *              TxString,
                              IN UINTN                TxStringLen
                            )
```

Send the HTTP message through TCP4.

Parameters:

[in]	*HttpInstance*	The HTTP instance private data.
[in]	*Wrap*	The HTTP token's wrap data.
[in]	*TxString*	Buffer containing the HTTP message string.
[in]	*TxStringLen*	Length of the HTTP message string in bytes.

Return values:

EFI_SUCCESS	The HTTP message is queued into TCP transmit queue.
Others	Other error as indicated.

Referenced by EfiHttpRequest(), and HttpTcpTransmit().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesDxe.c File Reference

Functions

EFI_STATUS EFIAPI HttpUtilitiesDxeUnload (IN EFI_HANDLE ImageHandle)
EFI_STATUS EFIAPI HttpUtilitiesDxeDriverEntryPoint (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)

Detailed Description

The DriverEntryPoint and Unload for HttpUtilities driver.

Copyright (c) 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI HttpUtilitiesDxeDriverEntryPoint (IN EFI_HANDLE *ImageHandle*, IN EFI_SYSTEM_TABLE * *SystemTable*)

This is the declaration of an EFI image entry point. This entry point is the same for UEFI Applications, UEFI OS Loaders, and UEFI Drivers including both device drivers and bus drivers.

Parameters:

ImageHandle The firmware allocated handle for the UEFI image.
SystemTable A pointer to the EFI System Table.

Return values:

EFI_SUCCESS The operation completed successfully.
Others An unexpected error occurred.

References mHttpUtilitiesProtocol.

EFI_STATUS EFIAPI HttpUtilitiesDxeUnload (IN EFI_HANDLE *ImageHandle*)

Unloads an image.

Parameters:

ImageHandle Handle that identifies the image to be unloaded.

Return values:

EFI_SUCCESS	The image has been unloaded.
EFI_INVALID_PARAMETER	ImageHandle is not a valid image handle.

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesDxe.h File Reference

Functions

VOID	FreeHeaderFields (IN EFI_HTTP_HEADER *HeaderFields, IN UINTN FieldCount)
EFI_HTTP_HEADER *	FindHttpHeader (IN EFI_HTTP_HEADER *HeaderFields, IN UINTN FieldCount, IN CHAR8 *FieldName)
BOOLEAN	IsValidHttpHeader (IN CHAR8 *DeleteList[], IN UINTN DeleteCount, IN CHAR8 *FieldName)
EFI_STATUS	SetFieldNameAndValue (IN EFI_HTTP_HEADER *HttpHeader, IN CHAR8 *FieldName, IN CHAR8 *FieldValue)
CHAR8 *	GetFieldNameAndValue (IN CHAR8 *String, OUT CHAR8 **FieldName, OUT CHAR8 **FieldValue)
EFI_STATUS EFIAPI	HttpUtilitiesBuild (IN EFI_HTTP_UTILITIES_PROTOCOL *This, IN UINTN SeedMessageSize, IN VOID *SeedMessage, IN UINTN DeleteCount, IN CHAR8 *DeleteList[], IN UINTN AppendCount, IN EFI_HTTP_HEADER *AppendList[], OUT UINTN *NewMessageSize, OUT VOID **NewMessage)
EFI_STATUS EFIAPI	HttpUtilitiesParse (IN EFI_HTTP_UTILITIES_PROTOCOL *This, IN CHAR8 *HttpMessage, IN UINTN HttpMessageSize, OUT EFI_HTTP_HEADER **HeaderFields, OUT UINTN *FieldCount)

Variables

EFI_HTTP_UTILITIES_PROTOCOL mHttpUtilitiesProtocol

Detailed Description

The header files of Http Utilities functions for HttpUtilities driver.

Copyright (c) 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_HTTP_HEADER* FindHttpHeader ( IN EFI_HTTP_HEADER *  HeaderFields,
                                  IN UINTN              FieldCount,
                                  IN CHAR8 *            FieldName
                                )
```

Find required header field in HeaderFields.

Parameters:

[in]	*HeaderFields*	Pointer to array of key/value header pairs.
[in]	*FieldCount*	The number of header pairs.
[in]	*FieldName*	Pointer to header field's name.

Returns:

Pointer to the queried header field.

NULL if not find this required header field.

Referenced by HttpUtilitiesBuild().

```
VOID FreeHeaderFields ( IN EFI_HTTP_HEADER *  HeaderFields,
                        IN UINTN              FieldCount
                      )
```

Free existing HeaderFields.

Parameters:

[in]	*HeaderFields*	Pointer to array of key/value header pairs waitting for free.
[in]	*FieldCount*	The number of header pairs in HeaderFields.

Referenced by HttpUtilitiesBuild(), and HttpUtilitiesParse().

```
CHAR8* GetFieldNameAndValue ( IN CHAR8 *      String,
                              OUT CHAR8 **    FieldName,
                              OUT CHAR8 **    FieldValue
                            )
```

Get one key/value header pair from the raw string.

Parameters:

[in]	*String*	Pointer to the raw string.
[out]	*FieldName*	Pointer to header field's name.
[out]	*FieldValue*	Pointer to header field's value.

Returns:

Pointer to the next raw string.

NULL if no key/value header pair from this raw string.

References AsciiStrGetNextToken().

Referenced by HttpUtilitiesParse().

```
EFI_STATUS EFIAPI
HttpUtilitiesBuild ( IN EFI_HTTP_UTILITIES_PROTOCOL *  This,
                     IN UINTN                          SeedMessageSize,
                     IN VOID *                         SeedMessage,
```

```
                                    IN UINTN                          DeleteCount,
                                    IN CHAR8 *                        DeleteList[],
                                    IN UINTN                          AppendCount,
                                    IN EFI_HTTP_HEADER *              AppendList[],
                                    OUT UINTN *                       NewMessageSize,
                                    OUT VOID **                       NewMessage
                                  )
```

Create HTTP header based on a combination of seed header, fields to delete, and fields to append.

The Build() function is used to manage the headers portion of an HTTP message by providing the ability to add, remove, or replace HTTP headers.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_UTILITIES_PROTOCOL instance.
[in]	*SeedMessageSize*	Size of the initial HTTP header. This can be zero.
[in]	*SeedMessage*	Initial HTTP header to be used as a base for building a new HTTP header. If NULL, SeedMessageSize is ignored.
[in]	*DeleteCount*	Number of null-terminated HTTP header field names in DeleteList.
[in]	*DeleteList*	List of null-terminated HTTP header field names to remove from SeedMessage. Only the field names are in this list because the field values are irrelevant to this operation.
[in]	*AppendCount*	Number of header fields in AppendList.
[in]	*AppendList*	List of HTTP headers to populate NewMessage with. If SeedMessage is not NULL, AppendList will be appended to the existing list from SeedMessage in NewMessage.
[out]	*NewMessageSize*	Pointer to number of header fields in NewMessage.
[out]	*NewMessage*	Pointer to a new list of HTTP headers based on.

Return values:

EFI_SUCCESS	Add, remove, and replace operations succeeded.
EFI_OUT_OF_RESOURCES	Could not allocate memory for NewMessage.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL.

References FindHttpHeader(), FreeHeaderFields(), IsValidHttpHeader(), and SetFieldNameAndValue().

```
EFI_STATUS EFIAPI HttpUtilitiesParse ( IN EFI_HTTP_UTILITIES_PROTOCOL *  This,
                                       IN CHAR8 *                        HttpMessage,
                                       IN UINTN                          HttpMessageSize,
                                       OUT EFI_HTTP_HEADER **            HeaderFields,
                                       OUT UINTN *                       FieldCount
                                     )
```

Parses HTTP header and produces an array of key/value pairs.

The Parse() function is used to transform data stored in HttpHeader into a list of fields paired with their corresponding values.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_UTILITIES_PROTOCOL instance.
[in]	*HttpMessage*	Contains raw unformatted HTTP header string.

[in]	*HttpMessageSize*	Size of HTTP header.
[out]	*HeaderFields*	Array of key/value header pairs.
[out]	*FieldCount*	Number of headers in HeaderFields.

Return values:

EFI_SUCCESS	Allocation succeeded.
EFI_NOT_STARTED	This EFI HTTP Protocol instance has not been initialized.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. HttpMessage is NULL. HeaderFields is NULL. FieldCount is NULL.

References FreeHeaderFields(), GetFieldNameAndValue(), and SetFieldNameAndValue().

```
BOOLEAN IsValidHttpHeader ( IN CHAR8 *  DeleteList[],
                            IN UINTN    DeleteCount,
                            IN CHAR8 *  FieldName
                          )
```

Check whether header field called FieldName is in DeleteList.

Parameters:

[in]	*DeleteList*	Pointer to array of key/value header pairs.
[in]	*DeleteCount*	The number of header pairs.
[in]	*FieldName*	Pointer to header field's name.

Returns:

TRUE if FieldName is not in DeleteList, that means this header field is valid.

FALSE if FieldName is in DeleteList, that means this header field is invalid.

Referenced by HttpUtilitiesBuild().

```
EFI_STATUS SetFieldNameAndValue ( IN EFI_HTTP_HEADER *  HttpHeader,
                                  IN CHAR8 *            FieldName,
                                  IN CHAR8 *            FieldValue
                                )
```

Set FieldName and FieldValue into specified HttpHeader.

Parameters:

[in]	*HttpHeader*	Specified HttpHeader.
[in]	*FieldName*	FieldName of this HttpHeader.
[in]	*FieldValue*	FieldValue of this HttpHeader.

Return values:

EFI_SUCCESS	The FieldName and FieldValue are set into HttpHeader successfully.
EFI_OUT_OF_RESOURCES	Failed to allocate resources.

Referenced by HttpUtilitiesBuild(), and HttpUtilitiesParse().

Variable Documentation

EFI_HTTP_UTILITIES_PROTOCOL mHttpUtilitiesProtocol

Referenced by HttpUtilitiesDxeDriverEntryPoint().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesImpl.c File Reference

Functions

CHAR8 *	AsciiStrGetNextToken (IN CONST CHAR8 *String, IN CHAR8 Seperator)
VOID	FreeHeaderFields (IN EFI_HTTP_HEADER *HeaderFields, IN UINTN FieldCount)
EFI_HTTP_HEADER *	FindHttpHeader (IN EFI_HTTP_HEADER *HeaderFields, IN UINTN FieldCount, IN CHAR8 *FieldName)
BOOLEAN	IsValidHttpHeader (IN CHAR8 *DeleteList[], IN UINTN DeleteCount, IN CHAR8 *FieldName)
EFI_STATUS	SetFieldNameAndValue (IN EFI_HTTP_HEADER *HttpHeader, IN CHAR8 *FieldName, IN CHAR8 *FieldValue)
CHAR8 *	GetFieldNameAndValue (IN CHAR8 *String, OUT CHAR8 **FieldName, OUT CHAR8 **FieldValue)

Detailed Description

The functions for HttpUtilities driver.

Copyright (c) 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
CHAR8* AsciiStrGetNextToken ( IN CONST CHAR8 *   String,
                              IN CHAR8           Seperator
                            )
```

Get the next string, which is distinguished by specified seperator.

Parameters:

[in]	*String*	Pointer to the string.
[in]	*Seperator*	Specified seperator used to distinguish where is the beginning of next string.

Returns:

Pointer to the next string.

NULL if not find or String is NULL.

Referenced by GetFieldNameAndValue().

```
EFI_HTTP_HEADER* FindHttpHeader ( IN EFI_HTTP_HEADER *   HeaderFields,
                                  IN UINTN               FieldCount,
                                  IN CHAR8 *             FieldName
                                )
```

Find required header field in HeaderFields.

Parameters:

[in]	*HeaderFields*	Pointer to array of key/value header pairs.
[in]	*FieldCount*	The number of header pairs.
[in]	*FieldName*	Pointer to header field's name.

Returns:

Pointer to the queried header field.

NULL if not find this required header field.

Referenced by HttpUtilitiesBuild().

```
VOID FreeHeaderFields ( IN EFI_HTTP_HEADER *   HeaderFields,
                        IN UINTN               FieldCount
                      )
```

Free existing HeaderFields.

Parameters:

[in]	*HeaderFields*	Pointer to array of key/value header pairs waitting for free.
[in]	*FieldCount*	The number of header pairs in HeaderFields.

Referenced by HttpUtilitiesBuild(), and HttpUtilitiesParse().

```
CHAR8* GetFieldNameAndValue ( IN CHAR8 *      String,
                              OUT CHAR8 **    FieldName,
                              OUT CHAR8 **    FieldValue
                            )
```

Get one key/value header pair from the raw string.

Parameters:

[in]	*String*	Pointer to the raw string.
[out]	*FieldName*	Pointer to header field's name.
[out]	*FieldValue*	Pointer to header field's value.

Returns:

Pointer to the next raw string.

NULL if no key/value header pair from this raw string.

References AsciiStrGetNextToken().

Referenced by HttpUtilitiesParse().

```
BOOLEAN IsValidHttpHeader ( IN CHAR8 *   DeleteList[],
                            IN UINTN     DeleteCount,
                            IN CHAR8 *   FieldName
                          )
```

Check whether header field called FieldName is in DeleteList.

Parameters:

[in]	*DeleteList*	Pointer to array of key/value header pairs.
[in]	*DeleteCount*	The number of header pairs.
[in]	*FieldName*	Pointer to header field's name.

Returns:

TRUE if FieldName is not in DeleteList, that means this header field is valid.

FALSE if FieldName is in DeleteList, that means this header field is invalid.

Referenced by HttpUtilitiesBuild().

```
EFI_STATUS SetFieldNameAndValue ( IN EFI_HTTP_HEADER *   HttpHeader,
                                  IN CHAR8 *             FieldName,
                                  IN CHAR8 *             FieldValue
                                )
```

Set FieldName and FieldValue into specified HttpHeader.

Parameters:

[in]	*HttpHeader*	Specified HttpHeader.
[in]	*FieldName*	FieldName of this HttpHeader.
[in]	*FieldValue*	FieldValue of this HttpHeader.

Return values:

EFI_SUCCESS	The FieldName and FieldValue are set into HttpHeader successfully.
EFI_OUT_OF_RESOURCES	Failed to allocate resources.

Referenced by HttpUtilitiesBuild(), and HttpUtilitiesParse().

NetworkPkg/HttpUtilitiesDxe/HttpUtilitiesProtocol.c File Reference

Functions

EFI_STATUS EFIAPI HttpUtilitiesBuild (IN EFI_HTTP_UTILITIES_PROTOCOL *This, IN UINTN SeedMessageSize, IN VOID *SeedMessage, IN UINTN DeleteCount, IN CHAR8 *DeleteList[], IN UINTN AppendCount, IN EFI_HTTP_HEADER *AppendList[], OUT UINTN *NewMessageSize, OUT VOID **NewMessage)

EFI_STATUS EFIAPI HttpUtilitiesParse (IN EFI_HTTP_UTILITIES_PROTOCOL *This, IN CHAR8 *HttpMessage, IN UINTN HttpMessageSize, OUT EFI_HTTP_HEADER **HeaderFields, OUT UINTN *FieldCount)

Variables

EFI_HTTP_UTILITIES_PROTOCOL mHttpUtilitiesProtocol

Detailed Description

Implementation of EFI_HTTP_PROTOCOL protocol interfaces.

Copyright (c) 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI
HttpUtilitiesBuild
                    ( IN EFI_HTTP_UTILITIES_PROTOCOL *   This,
                      IN UINTN                           SeedMessageSize,
                      IN VOID *                          SeedMessage,
                      IN UINTN                           DeleteCount,
                      IN CHAR8 *                         DeleteList[],
                      IN UINTN                           AppendCount,
                      IN EFI_HTTP_HEADER *               AppendList[],
                      OUT UINTN *                        NewMessageSize,
                      OUT VOID **                        NewMessage
                    )
```

Create HTTP header based on a combination of seed header, fields to delete, and fields to append.

The Build() function is used to manage the headers portion of an HTTP message by providing the ability to add, remove, or replace HTTP headers.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_UTILITIES_PROTOCOL instance.
[in]	*SeedMessageSize*	Size of the initial HTTP header. This can be zero.
[in]	*SeedMessage*	Initial HTTP header to be used as a base for building a new HTTP header. If NULL, SeedMessageSize is ignored.
[in]	*DeleteCount*	Number of null-terminated HTTP header field names in DeleteList.
[in]	*DeleteList*	List of null-terminated HTTP header field names to remove from SeedMessage. Only the field names are in this list because the field values are irrelevant to this operation.
[in]	*AppendCount*	Number of header fields in AppendList.
[in]	*AppendList*	List of HTTP headers to populate NewMessage with. If SeedMessage is not NULL, AppendList will be appended to the existing list from SeedMessage in NewMessage.
[out]	*NewMessageSize*	Pointer to number of header fields in NewMessage.
[out]	*NewMessage*	Pointer to a new list of HTTP headers based on.

Return values:

EFI_SUCCESS	Add, remove, and replace operations succeeded.
EFI_OUT_OF_RESOURCES	Could not allocate memory for NewMessage.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL.

References FindHttpHeader(), FreeHeaderFields(), IsValidHttpHeader(), and SetFieldNameAndValue().

```
EFI_STATUS EFIAPI HttpUtilitiesParse ( IN EFI_HTTP_UTILITIES_PROTOCOL *  This,
                                       IN CHAR8 *                        HttpMessage,
                                       IN UINTN                          HttpMessageSize,
                                       OUT EFI_HTTP_HEADER **            HeaderFields,
                                       OUT UINTN *                       FieldCount
                                     )
```

Parses HTTP header and produces an array of key/value pairs.

The Parse() function is used to transform data stored in HttpHeader into a list of fields paired with their corresponding values.

Parameters:

[in]	*This*	Pointer to EFI_HTTP_UTILITIES_PROTOCOL instance.
[in]	*HttpMessage*	Contains raw unformatted HTTP header string.
[in]	*HttpMessageSize*	Size of HTTP header.
[out]	*HeaderFields*	Array of key/value header pairs.
[out]	*FieldCount*	Number of headers in HeaderFields.

Return values:

EFI_SUCCESS	Allocation succeeded.
EFI_NOT_STARTED	This EFI HTTP Protocol instance has not been initialized.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. HttpMessage is NULL. HeaderFields is NULL. FieldCount is NULL.

References FreeHeaderFields(), GetFieldNameAndValue(), and SetFieldNameAndValue().

Variable Documentation

EFI_HTTP_UTILITIES_PROTOCOL mHttpUtilitiesProtocol

Initial value:

```
{
  HttpUtilitiesBuild,
  HttpUtilitiesParse
}
```

Referenced by HttpUtilitiesDxeDriverEntryPoint().

NetworkPkg/Include/Guid/Ip6ConfigHii.h File Reference

Defines

"memItemRight" valign="bottom">IP6_CONFIG_NVDATA_GUID

Variables

EFI_GUID gIp6ConfigNvDataGuid

Detailed Description

GUIDs used as HII FormSet and HII Package list GUID in Ip6Config driver.

Copyright (c) 2011, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Value:

```
{ \
    0x2eea107, 0x98db, 0x400e, { 0x98, 0x30, 0x46, 0xa, 0x15, 0x42, 0xd7, 0x99 } \
  }
```

Variable Documentation

EFI_GUID gIp6ConfigNvDataGuid

Referenced by Ip6ConfigFormInit(), Ip6ConvertAddressListToString(), Ip6FormExtractConfig(), and Ip6FormRouteConfig().

- File List
- Globals

NetworkPkg/Include/Guid/IScsiConfigHii.h File Reference

Defines

"memItemRight" valign="bottom">ISCSI_CONFIG_GUID

Variables

EFI_GUID gIScsiConfigGuid

Detailed Description

GUIDs used as HII FormSet and HII Package list GUID in IScsiConfig driver that supports IP4 and IP6 both.

Copyright (c) 2011, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License that accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Value:

```
{ \
    0x4b47d616, 0xa8d6, 0x4552, { 0x9d, 0x44, 0xcc, 0xad, 0x2e, 0xf, 0x4c, 0xf9 } \
  }
```

Variable Documentation

EFI_GUID gIScsiConfigGuid

Referenced by IScsiConfigAddAttempt(), IScsiConfigDeleteAttempts(), IScsiConfigDisplayDeleteAttempts(), IScsiConfigDisplayOrderAttempts(), IScsiConfigFormInit(), IScsiConfigOrderAttempts(), IScsiConfigProcessDefault(), IScsiConfigUpdateAttempt(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiDhcpIsConfigured(), IScsiFormExtractConfig(), IScsiFormRouteConfig(), IScsiGetConfigData(), and IScsiStart().

- Data Structures
- Files

- File List
- Globals

NetworkPkg/Ip6Dxe/ComponentName.c File Reference

Functions

EFI_STATUS EFIAPI Ip6ComponentNameGetDriverName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN CHAR8 *Language, OUT CHAR16 **DriverName)

EFI_STATUS EFIAPI Ip6ComponentNameGetControllerName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE ChildHandle, IN CHAR8 *Language, OUT CHAR16 **ControllerName)

EFI_STATUS UpdateName (IN EFI_IP6_PROTOCOL *Ip6)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gIp6ComponentName

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gIp6ComponentName2

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mIp6DriverNameTable []

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE * gIp6ControllerNameTable = NULL

Detailed Description

Implementation of EFI_COMPONENT_NAME_PROTOCOL and EFI_COMPONENT_NAME2_PROTOCOL protocol.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI Ip6ComponentNameGetControllerName	(IN EFI_COMPONENT_NAME_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_HANDLE	*ChildHandle*,

	IN CHAR8 *	*Language*,
	OUT CHAR16 **	*ControllerName*
)		

Retrieves a Unicode string that is the user-readable name of the controller that is managed by a driver.

This function retrieves the user-readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

`[in]`	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
`[in]`	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
`[in]`	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
`[in]`	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
`[out]`	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL, and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

Retrieves a Unicode string that is the user-readable name of the controller that is being managed by a driver.

This function retrieves the user-readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
[in]	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL, and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gIp6ComponentName, gIp6ControllerNameTable, and UpdateName().

```
EFI_STATUS EFIAPI
Ip6ComponentNameGetDriverName (
  IN EFI_COMPONENT_NAME_PROTOCOL *  This,
  IN CHAR8 *                        Language,
  OUT CHAR16 **                     DriverName
)
```

Retrieves a Unicode string that is the user-readable name of the driver.

This function retrieves the user-readable name of a driver in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the driver name is returned in DriverName, and EFI_SUCCESS is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This

is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.

`[out]` *DriverName* A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by This in the language specified by Language.

Return values:

EFI_SUCCESS	The Unicode string for the Driver specified by This and the language specified by Language was returned in DriverName.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	DriverName is NULL.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gIp6ComponentName, and mIp6DriverNameTable.

EFI_STATUS UpdateName (IN EFI_IP6_PROTOCOL * *Ip6*)

Update the component name for the IP6 child handle.

Parameters:

Ip6[in] A pointer to the EFI_IP6_PROTOCOL.

Return values:

EFI_SUCCESS	Update the ControllerNameTable of this instance successfully.
EFI_INVALID_PARAMETER	The input parameter is invalid.

References gIp6ComponentName, gIp6ComponentName2, and gIp6ControllerNameTable.

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gIp6ComponentName

Initial value:

```
{
 Ip6ComponentNameGetDriverName,
 Ip6ComponentNameGetControllerName,
 "eng"
}
```

Referenced by Ip6ComponentNameGetControllerName(), Ip6ComponentNameGetDriverName(), Ip6DriverEntryPoint(), and UpdateName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gIp6ComponentName2

Initial value:

```
{
 (EFI_COMPONENT_NAME2_GET_DRIVER_NAME) Ip6ComponentNameGetDriverName,
 (EFI_COMPONENT_NAME2_GET_CONTROLLER_NAME) Ip6ComponentNameGetControllerName,
 "en"
}
```

Referenced by Ip6DriverEntryPoint(), and UpdateName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE* gIp6ControllerNameTable = NULL

Referenced by Ip6ComponentNameGetControllerName(), and UpdateName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mIp6DriverNameTable[]

Initial value:

```
 {
  {
    "eng;en",
    L"IP6 Network Service Driver"
  },
  {
    NULL,
    NULL
  }
}
```

Referenced by Ip6ComponentNameGetDriverName().

NetworkPkg/Ip6Dxe/Ip6Common.c File Reference

Functions

EFI_STATUS Ip6BuildEfiAddressList (IN IP6_SERVICE *IpSb, OUT UINT32 *AddressCount, OUT EFI_IP6_ADDRESS_INFO **AddressList)
EFI_STATUS Ip6SetToAllNodeMulticast (IN BOOLEAN Router, IN UINT8 Scope, OUT EFI_IPv6_ADDRESS *Ip6Addr)
UINT8 * Ip6CreateInterfaceID (IN OUT IP6_SERVICE *IpSb)
EFI_IPv6_ADDRESS * Ip6CreateLinkLocalAddr (IN OUT IP6_SERVICE *IpSb)
VOID Ip6CreateSNMulticastAddr (IN EFI_IPv6_ADDRESS *Ip6Addr, OUT EFI_IPv6_ADDRESS *MulticastAddr)
VOID Ip6AddAddr (IN OUT IP6_INTERFACE *IpIf, IN IP6_ADDRESS_INFO *AddrInfo)
EFI_STATUS EFIAPI Ip6DestroyChildEntryByAddr (IN LIST_ENTRY *Entry, IN VOID *Context)
VOID Ip6DestroyInstanceByAddress (IN OUT IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *Address)
EFI_STATUS Ip6RemoveAddr (IN IP6_SERVICE *IpSb, IN OUT LIST_ENTRY *AddressList, IN OUT UINT32 *AddressCount, IN EFI_IPv6_ADDRESS *Prefix, IN UINT8 PrefixLength)
BOOLEAN Ip6IsSNMulticastAddr (IN EFI_IPv6_ADDRESS *Ip6)
BOOLEAN Ip6IsOneOfSetAddress (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *Address, OUT IP6_INTERFACE **Interface, OUT IP6_ADDRESS_INFO **AddressInfo)
BOOLEAN Ip6IsValidLinkAddress (IN IP6_SERVICE *IpSb, IN EFI_MAC_ADDRESS *LinkAddress)
VOID Ip6CopyAddressByPrefix (OUT EFI_IPv6_ADDRESS *Dest, IN EFI_IPv6_ADDRESS *Src, IN UINT8 PrefixLength)
EFI_STATUS Ip6GetMulticastMac (IN EFI_MANAGED_NETWORK_PROTOCOL *Mnp, IN EFI_IPv6_ADDRESS *Multicast, OUT EFI_MAC_ADDRESS *Mac)
EFI_IP6_HEADER * Ip6NtohHead (IN OUT EFI_IP6_HEADER *Head)

Detailed Description

The implementation of common functions shared by IP6 driver.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

VOID Ip6AddAddr (IN OUT IP6_INTERFACE * *IpIf*,

```
                    IN IP6_ADDRESS_INFO *     AddrInfo
                  )
```

Insert a node IP6_ADDRESS_INFO to an IP6 interface.

Parameters:

[in,out]	*IpIf*	Points to an IP6 interface.
[in]	*AddrInfo*	Points to IP6_ADDRESS_INFO

Referenced by Ip6OnDADFinished().

```
EFI_STATUS Ip6BuildEfiAddressList ( IN IP6_SERVICE *                  IpSb,
                                    OUT UINT32 *                      AddressCount,
                                    OUT EFI_IP6_ADDRESS_INFO **       AddressList
                                  )
```

Build a array of EFI_IP6_ADDRESS_INFO to be returned to the caller. The number of EFI_IP6_ADDRESS_INFO is also returned. If AddressList is NULL, only the address count is returned.

Parameters:

[in]	*IpSb*	The IP6 service binding instance.
[out]	*AddressCount*	The number of returned addresses.
[out]	*AddressList*	The pointer to the array of EFI_IP6_ADDRESS_INFO. This is an optional parameter.

Return values:

EFI_SUCCESS	The address array successfully built.
EFI_OUT_OF_RESOURCES	Failed to allocate the memory for the address info.
EFI_INVALID_PARAMETER	Any input parameter is invalid.

References _IP6_ADDRESS_INFO::Address, IP6_ADDR_INFO_SIGNATURE, IP6_LINK_LOCAL_PREFIX_LENGTH, and _IP6_ADDRESS_INFO::PrefixLength.

Referenced by EfiIp6GetModeData(), and Ip6ConfigGetIfInfo().

```
VOID Ip6CopyAddressByPrefix ( OUT EFI_IPv6_ADDRESS *  Dest,
                              IN EFI_IPv6_ADDRESS *   Src,
                              IN UINT8                PrefixLength
                            )
```

Copy the PrefixLength bits from Src to Dest.

Parameters:

[out]	*Dest*	A pointer to the buffer to copy to.
[in]	*Src*	A pointer to the buffer to copy from.
[in]	*PrefixLength*	The number of bits to copy.

Referenced by Ip6BuildEfiRouteTable().

```
UINT8* Ip6CreateInterfaceID ( IN OUT IP6_SERVICE *   IpSb   )
```

This function converts MAC address to 64 bits interface ID according to RFC4291 and returns the interface

ID. Currently only 48-bit MAC address is supported by this function.

Parameters:

`[in,out]` *IpSb* The IP6 service binding instance.

Return values:

NULL The operation fails.

Returns:

Pointer to the generated interface ID.

References IP6_IF_ID_LEN, IP6_MAC_LEN, IP6_SERVICE_SIGNATURE, and IP6_U_BIT.

Referenced by Ip6CreateLinkLocalAddr().

EFI_IPv6_ADDRESS* Ip6CreateLinkLocalAddr (IN OUT IP6_SERVICE * *IpSb*)

This function creates link-local address from interface identifier. The interface identifier is normally created from MAC address. It might be manually configured by administrator if the link-local address created from MAC address is a duplicate address.

Parameters:

`[in,out]` *IpSb* The IP6 service binding instance.

Return values:

NULL If the operation fails.

Returns:

The generated Link Local address, in network order.

References IP6_SERVICE_SIGNATURE, and Ip6CreateInterfaceID().

Referenced by Ip6CreateInterface().

```
VOID Ip6CreateSNMulticastAddr ( IN EFI_IPv6_ADDRESS *     Ip6Addr,
                                OUT EFI_IPv6_ADDRESS *    MulticastAddr
                              )
```

Compute the solicited-node multicast address for an unicast or anycast address, by taking the low-order 24 bits of this address, and appending those bits to the prefix FF02:0:0:0:0:1:FF00::/104.

Parameters:

`[in]` *Ip6Addr* The unicast or anycast address, in network order.

`[out]` *MulticastAddr* The generated solicited-node multicast address, in network order.

Referenced by Ip6InitDADProcess(), Ip6IsSNMulticastAddr(), Ip6NdFasterTimerTicking(), Ip6Output(), and Ip6RemoveAddr().

```
EFI_STATUS EFIAPI Ip6DestroyChildEntryByAddr ( IN LIST_ENTRY *  Entry,
                                               IN VOID *        Context
                                             )
```

Callback function which provided by user to remove one node in NetDestroyLinkList process.

Parameters:

[in]	*Entry*	The entry to be removed.
[in]	*Context*	Pointer to the callback context corresponds to the Context in NetDestroyLinkList.

Return values:

EFI_SUCCESS	The entry has been removed successfully.
Others	Fail to remove the entry.

References _IP6_PROTOCOL::ConfigData, _IP6_PROTOCOL::Handle, IP6_PROTOCOL_SIGNATURE, IP6_STATE_CONFIGED, and _IP6_PROTOCOL::State.

Referenced by Ip6DestroyInstanceByAddress().

```
VOID Ip6DestroyInstanceByAddress ( IN OUT IP6_SERVICE * IpSb,
                                   IN EFI_IPv6_ADDRESS * Address
                                 )
```

Destroy the IP instance if its StationAddress is removed. It is the help function for Ip6RemoveAddr().

Parameters:

[in,out]	*IpSb*	Points to an IP6 service binding instance.
[in]	*Address*	The to be removed address

References IP6_DESTROY_CHILD_BY_ADDR_CALLBACK_CONTEXT::Address, IP6_SERVICE_SIGNATURE, Ip6DestroyChildEntryByAddr(), and IP6_DESTROY_CHILD_BY_ADDR_CALLBACK_CONTEXT::ServiceBinding.

Referenced by Ip6RemoveAddr().

```
EFI_STATUS Ip6GetMulticastMac ( IN EFI_MANAGED_NETWORK_PROTOCOL * Mnp,
                                IN EFI_IPv6_ADDRESS * Multicast,
                                OUT EFI_MAC_ADDRESS * Mac
                              )
```

Get the MAC address for a multicast IP address. Call Mnp's McastIpToMac to find the MAC address instead of hard-coding the NIC to be Ethernet.

Parameters:

[in]	*Mnp*	The Mnp instance to get the MAC address.
[in]	*Multicast*	The multicast IP address to translate.
[out]	*Mac*	The buffer to hold the translated address.

Return values:

EFI_SUCCESS	The multicast IP successfully translated to a multicast MAC address.
Other	The address is not converted because an error occurred.

Referenced by Ip6InitMld(), Ip6JoinGroup(), and Ip6SendFrame().

```
BOOLEAN Ip6IsOneOfSetAddress ( IN IP6_SERVICE *              IpSb,
                               IN EFI_IPv6_ADDRESS *         Address,
                               OUT IP6_INTERFACE **          Interface,
                               OUT IP6_ADDRESS_INFO **       AddressInfo
                             )
```

Check whether the incoming IPv6 address is one of the maintained addresses in the IP6 service binding instance.

Parameters:

[in]	*IpSb*	Points to a IP6 service binding instance.
[in]	*Address*	The IP6 address to be checked.
[out]	*Interface*	If not NULL, output the IP6 interface which maintains the Address.
[out]	*AddressInfo*	If not NULL, output the IP6 address information of the Address.

Return values:

TRUE	Yes, it is one of the maintained address.
FALSE	No, it is not one of the maintained address.

References _IP6_ADDRESS_INFO::Address, _IP6_INTERFACE::AddressList, IP6_ADDR_INFO_SIGNATURE, and IP6_INTERFACE_SIGNATURE.

Referenced by EfiIp6Neighbors(), Ip6ConfigOnDhcp6Event(), Ip6ConfigProtocol(), Ip6ConfigSetMaunualAddress(), Ip6IcmpReplyEcho(), Ip6IsValidAddress(), Ip6PreProcessPacket(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), Ip6ProcessRouterAdvertise(), Ip6SelectInterface(), Ip6SelectSourceAddress(), and Ip6SetAddress().

```
BOOLEAN Ip6IsSNMulticastAddr ( IN EFI_IPv6_ADDRESS * Ip6 )
```

Check whether the incoming Ipv6 address is a solicited-node multicast address.

Parameters:

[in]	*Ip6*	Ip6 address, in network order.

Return values:

TRUE	Yes, solicited-node multicast address
FALSE	No

References Ip6CreateSNMulticastAddr().

Referenced by Ip6ProcessNeighborSolicit(), and Ip6SendNeighborSolicit().

```
BOOLEAN Ip6IsValidLinkAddress ( IN IP6_SERVICE *          IpSb,
                                IN EFI_MAC_ADDRESS *      LinkAddress
                              )
```

Check whether the incoming MAC address is valid.

Parameters:

[in]	*IpSb*	Points to a IP6 service binding instance.
[in]	*LinkAddress*	The MAC address.

Return values:

TRUE Yes, it is valid.
FALSE No, it is not valid.

Referenced by EfiIp6Neighbors().

EFI_IP6_HEADER* Ip6NtohHead (IN OUT EFI_IP6_HEADER * *Head*)

Convert the multibyte field in IP header's byter order. In spite of its name, it can also be used to convert from host to network byte order.

Parameters:

`[in,out]` *Head* The IP head to convert.

Returns:

Point to the converted IP head.

Referenced by Ip6IpSecProcessPacket(), Ip6PreProcessPacket(), Ip6Reassemble(), and Ip6WrapRxData().

```
EFI_STATUS Ip6RemoveAddr ( IN IP6_SERVICE *            IpSb,
                           IN OUT LIST_ENTRY *         AddressList,
                           IN OUT UINT32 *             AddressCount,
                           IN EFI_IPv6_ADDRESS *       Prefix,
                           IN UINT8                    PrefixLength
                         )
```

Remove the IPv6 address from the address list node points to IP6_ADDRESS_INFO.

This function removes the matching IPv6 addresses from the address list and adjusts the address count of the address list. If IpSb is not NULL, this function calls Ip6LeaveGroup to see whether it should call Mnp->Groups() to remove the its solicited-node multicast MAC address from the filter list and sends out a Multicast Listener Done. If Prefix is NULL, all address in the address list will be removed. If Prefix is not NULL, the address that matching the Prefix with PrefixLength in the address list will be removed.

Parameters:

`[in]`	*IpSb*	NULL or points to IP6 service binding instance.
`[in,out]`	*AddressList*	Address list array.
`[in,out]`	*AddressCount*	The count of addresses in address list array.
`[in]`	*Prefix*	NULL or an IPv6 address prefix.
`[in]`	*PrefixLength*	The length of Prefix.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_FOUND	The address matching the Prefix with PrefixLength cannot be found in the address list.
EFI_INVALID_PARAMETER	Any input parameter is invalid.

References _IP6_ADDRESS_INFO::Address, IP6_ADDR_INFO_SIGNATURE, IP6_SERVICE_SIGNATURE, Ip6CreateSNMulticastAddr(), Ip6DestroyInstanceByAddress(), Ip6LeaveGroup(), and _IP6_ADDRESS_INFO::PrefixLength.

Referenced by Ip6CleanInterface(), Ip6ConfigOnDhcp6Event(), Ip6ConfigOnPolicyChanged(), Ip6ConfigSetMaunualAddress(), Ip6DestroyPrefixListEntry(), and Ip6SelectSourceAddress().

```
EFI_STATUS Ip6SetToAllNodeMulticast ( IN BOOLEAN                Router,
                                      IN UINT8                  Scope,
                                      OUT EFI_IPv6_ADDRESS *    Ip6Addr
                                    )
```

Generate the multicast addresses identify the group of all IPv6 nodes or IPv6 routers defined in RFC4291.

All Nodes Addresses: FF01::1, FF02::1. All Router Addresses: FF01::2, FF02::2, FF05::2.

Parameters:

[in]	*Router*	If TRUE, generate all routers addresses, else generate all node addresses.
[in]	*Scope*	interface-local(1), link-local(2), or site-local(5)
[out]	*Ip6Addr*	The generated multicast address.

Return values:

EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_SUCCESS	The address is generated.

References IP6_SITE_LOCAL_SCOPE.

Referenced by Ip6CleanService(), Ip6InitMld(), Ip6OnDADFinished(), Ip6ProcessMldQuery(), Ip6ProcessNeighborSolicit(), Ip6SendMldDone(), and Ip6SendRouterSolicit().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Ip6Dxe/Ip6Common.h File Reference

Data Structures

Defines

"memItemRight" valign="bottom">IP6_LINK_EQUAL(Mac1, Mac2) (CompareMem ((Mac1), (Mac2), sizeof (EF

Typedefs

typede
typed
typ
typedef str

Enumerations

Functions

E

Detailed Description

Common definition and functions for IP6 driver.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Ip6CreateLinkTxToken(), and Ip6CreateService().

#define IP6_HOP_LIMIT 255

Referenced by Ip6CreateService(), Ip6ProcessNeighborAdvertise(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), Ip6ProcessRouterAdvertise(), Ip6SendNeighborAdvertise(), Ip6SendNeighborSolicit(), and Ip6SendRouterSolicit().

#define IP6_IF_ID_LEN 8

Referenced by Ip6CreateInterfaceID(), and Ip6CreateService().

#define IP6_INFINIT_LIFETIME 0xFFFFFFFF

Referenced by Ip6AddNeighbor(), Ip6CandidateSource(), Ip6ConfigOnPolicyChanged(), Ip6ConfigSetMaunualAddress(), Ip6CreateInterface(), Ip6InitMld(), Ip6NdFasterTimerTicking(), Ip6NdTimerTicking(), Ip6OnDADFinished(), Ip6ProcessNeighborAdvertise(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), and Ip6ProcessRouterAdvertise().

```
#define IP6_INTERFACE_LOCAL_SCOPE   1
```

```
#define IP6_LINK_BROADCAST   0x00000001
```

Referenced by Ip6OnFrameReceivedDpc().

```
#define
IP6_LINK_EQUAL          ( Mac1,
                          Mac2    )   (CompareMem ((Mac1), (Mac2), sizeof (EFI_MAC_ADDRESS))
                                      == 0)
```

```
#define IP6_LINK_LOCAL_PREFIX_LENGTH   64
```

Referenced by Ip6BuildEfiAddressList(), Ip6CandidateSource(), Ip6ConfigOnPolicyChanged(), Ip6ConfigProtocol(), Ip6CreateInterface(), Ip6OnDADFinished(), Ip6ProcessRouterAdvertise(), and Ip6SetAddress().

```
#define IP6_LINK_LOCAL_SCOPE   2
```

Referenced by Ip6CleanService(), Ip6InitMld(), Ip6OnDADFinished(), Ip6ProcessMldQuery(), Ip6ProcessNeighborSolicit(), Ip6SendMldDone(), and Ip6SendRouterSolicit().

```
#define IP6_LINK_MULTICAST   0x00000002
```

Referenced by Ip6OnFrameReceivedDpc().

```
#define IP6_LINK_PROMISC   0x00000004
```

Referenced by Ip6OnFrameReceivedDpc().

```
#define IP6_MAC_LEN   6
```

Referenced by Ip6CreateInterfaceID().

```
#define IP6_ONE_SECOND_IN_MS   1000
```

Referenced by Ip6ConfigOnPolicyChanged(), Ip6DriverBindingStart(), and Ip6SetAddress().

#define IP6_SITE_LOCAL_SCOPE 5

Referenced by Ip6SetToAllNodeMulticast().

#define IP6_TIMER_INTERVAL_IN_MS 100

Referenced by Ip6DriverBindingStart(), Ip6InitDADProcess(), and Ip6SetAddress().

#define IP6_U_BIT 0x02

Referenced by Ip6CreateInterfaceID().

#define IP6_US_TO_SEC (Us) (((Us) + 999999) / 1000000)

Referenced by EfiIp6Transmit(), and Ip6InstanceEnquePacket().

Typedef Documentation

typedef struct _IP6_ADDRESS_INFO IP6_ADDRESS_INFO

typedef struct _IP6_INTERFACE IP6_INTERFACE

typedef struct _IP6_PROTOCOL IP6_PROTOCOL

typedef struct _IP6_SERVICE IP6_SERVICE

Enumeration Type Documentation

enum IP6_ADDRESS_TYPE

Enumerator:

Ip6Promiscuous
Ip6Unicast
Ip6Multicast
Ip6AnyCast

Function Documentation

VOID Ip6AddAddr (IN OUT IP6_INTERFACE * *IpIf*,
IN IP6_ADDRESS_INFO * *AddrInfo*

)

Insert a node IP6_ADDRESS_INFO to an IP6 interface.

Parameters:

[in,out]	*IpIf*	Points to an IP6 interface.
[in]	*AddrInfo*	Points to an IP6_ADDRESS_INFO.

Insert a node IP6_ADDRESS_INFO to an IP6 interface.

Parameters:

[in,out]	*IpIf*	Points to an IP6 interface.
[in]	*AddrInfo*	Points to IP6_ADDRESS_INFO

Referenced by Ip6OnDADFinished().

```
EFI_STATUS Ip6BuildEfiAddressList ( IN IP6_SERVICE *                IpSb,
                                    OUT UINT32 *                    AddressCount,
                                    OUT EFI_IP6_ADDRESS_INFO **     AddressList
                                  )
```

Build a array of EFI_IP6_ADDRESS_INFO to be returned to the caller. The number of EFI_IP6_ADDRESS_INFO is also returned. If AddressList is NULL, only the address count is returned.

Parameters:

[in]	*IpSb*	The IP6 service binding instance.
[out]	*AddressCount*	The number of returned addresses.
[out]	*AddressList*	The pointer to the array of EFI_IP6_ADDRESS_INFO. This is an optional parameter.

Return values:

EFI_SUCCESS	The address array is successfully build
EFI_OUT_OF_RESOURCES	Failed to allocate the memory for the address info.
EFI_INVALID_PARAMETER	Any input parameter is invalid.

Build a array of EFI_IP6_ADDRESS_INFO to be returned to the caller. The number of EFI_IP6_ADDRESS_INFO is also returned. If AddressList is NULL, only the address count is returned.

Parameters:

[in]	*IpSb*	The IP6 service binding instance.
[out]	*AddressCount*	The number of returned addresses.
[out]	*AddressList*	The pointer to the array of EFI_IP6_ADDRESS_INFO. This is an optional parameter.

Return values:

EFI_SUCCESS	The address array successfully built.
EFI_OUT_OF_RESOURCES	Failed to allocate the memory for the address info.
EFI_INVALID_PARAMETER	Any input parameter is invalid.

References _IP6_ADDRESS_INFO::Address, IP6_ADDR_INFO_SIGNATURE, IP6_LINK_LOCAL_PREFIX_LENGTH, and _IP6_ADDRESS_INFO::PrefixLength.

Referenced by EfiIp6GetModeData(), and Ip6ConfigGetIfInfo().

```
VOID Ip6CopyAddressByPrefix ( OUT EFI_IPv6_ADDRESS *  Dest,
                              IN EFI_IPv6_ADDRESS *   Src,
                              IN UINT8                PrefixLength
                            )
```

Copy the PrefixLength bits from Src to Dest.

Parameters:

[out]	*Dest*	A pointer to the buffer to copy to.
[in]	*Src*	A pointer to the buffer to copy from.
[in]	*PrefixLength*	The number of bits to copy.

Referenced by Ip6BuildEfiRouteTable().

```
UINT8* Ip6CreateInterfaceID ( IN OUT IP6_SERVICE *  IpSb  )
```

This function converts MAC address to 64 bits interface ID according to RFC4291 and returns the interface ID. Currently only 48-bit MAC address is supported by this function.

Parameters:

[in,out] *IpSb* The IP6 service binding instance.

Return values:

NULL The operation fails.

Returns:

Pointer to the generated interface ID.

References IP6_IF_ID_LEN, IP6_MAC_LEN, IP6_SERVICE_SIGNATURE, and IP6_U_BIT.

Referenced by Ip6CreateLinkLocalAddr().

```
EFI_IPv6_ADDRESS* Ip6CreateLinkLocalAddr ( IN OUT IP6_SERVICE *  IpSb  )
```

This function creates link-local address from interface identifier. The interface identifier is normally created from MAC address. It might be manually configured by administrator if the link-local address created from MAC address is a duplicate address.

Parameters:

[in,out] *IpSb* The IP6 service binding instance.

Return values:

NULL If the operation fails.

Returns:

The generated Link Local address, in network order.

References IP6_SERVICE_SIGNATURE, and Ip6CreateInterfaceID().

Referenced by Ip6CreateInterface().

```
VOID Ip6CreateSNMulticastAddr ( IN EFI_IPv6_ADDRESS *   Ip6Addr,
                                OUT EFI_IPv6_ADDRESS *  MulticastAddr
```

)

Compute the solicited-node multicast address for an unicast or anycast address, by taking the low-order 24 bits of this address, and appending those bits to the prefix FF02:0:0:0:0:1:FF00::/104.

Parameters:

Ip6Addr The unicast or anycast address, in network order.
MulticastAddr The generated solicited-node multicast address, in network order.

Compute the solicited-node multicast address for an unicast or anycast address, by taking the low-order 24 bits of this address, and appending those bits to the prefix FF02:0:0:0:0:1:FF00::/104.

Parameters:

`[in]` *Ip6Addr* The unicast or anycast address, in network order.
`[out]` *MulticastAddr* The generated solicited-node multicast address, in network order.

Referenced by Ip6InitDADProcess(), Ip6IsSNMulticastAddr(), Ip6NdFasterTimerTicking(), Ip6Output(), and Ip6RemoveAddr().

```
EFI_STATUS Ip6GetMulticastMac ( IN EFI_MANAGED_NETWORK_PROTOCOL *  Mnp,
                                IN EFI_IPv6_ADDRESS *              Multicast,
                                OUT EFI_MAC_ADDRESS *              Mac
                              )
```

Get the MAC address for a multicast IP address. Call Mnp's McastIpToMac to find the MAC address instead of hard-coding the NIC to be Ethernet.

Parameters:

`[in]` *Mnp* The Mnp instance to get the MAC address.
`[in]` *Multicast* The multicast IP address to translate.
`[out]` *Mac* The buffer to hold the translated address.

Return values:

EFI_SUCCESS The multicast IP is successfully translated to a multicast MAC address.
Other The address is not converted because an error occurred.

Get the MAC address for a multicast IP address. Call Mnp's McastIpToMac to find the MAC address instead of hard-coding the NIC to be Ethernet.

Parameters:

`[in]` *Mnp* The Mnp instance to get the MAC address.
`[in]` *Multicast* The multicast IP address to translate.
`[out]` *Mac* The buffer to hold the translated address.

Return values:

EFI_SUCCESS The multicast IP successfully translated to a multicast MAC address.
Other The address is not converted because an error occurred.

Referenced by Ip6InitMld(), Ip6JoinGroup(), and Ip6SendFrame().

```
BOOLEAN Ip6IsOneOfSetAddress ( IN IP6_SERVICE *          IpSb,
                               IN EFI_IPv6_ADDRESS *     Address,
```

```
                                    OUT IP6_INTERFACE **        Interface,
                                    OUT IP6_ADDRESS_INFO **     AddressInfo
                                  )
```

Check whether the incoming IPv6 address is one of the maintained address in the IP6 service binding instance.

Parameters:

[in]	*IpSb*	Points to a IP6 service binding instance
[in]	*Address*	The IP6 address to be checked.
[out]	*Interface*	If not NULL, output the IP6 interface which maintains the Address.
[out]	*AddressInfo*	If not NULL, output the IP6 address information of the Address.

Return values:

TRUE	Yes, it is one of the maintained addresses.
FALSE	No, it is not one of the maintained addresses.

Check whether the incoming IPv6 address is one of the maintained addresses in the IP6 service binding instance.

Parameters:

[in]	*IpSb*	Points to a IP6 service binding instance.
[in]	*Address*	The IP6 address to be checked.
[out]	*Interface*	If not NULL, output the IP6 interface which maintains the Address.
[out]	*AddressInfo*	If not NULL, output the IP6 address information of the Address.

Return values:

TRUE	Yes, it is one of the maintained address.
FALSE	No, it is not one of the maintained address.

References _IP6_ADDRESS_INFO::Address, _IP6_INTERFACE::AddressList, IP6_ADDR_INFO_SIGNATURE, and IP6_INTERFACE_SIGNATURE.

Referenced by EfiIp6Neighbors(), Ip6ConfigOnDhcp6Event(), Ip6ConfigProtocol(), Ip6ConfigSetMaunualAddress(), Ip6IcmpReplyEcho(), Ip6IsValidAddress(), Ip6PreProcessPacket(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), Ip6ProcessRouterAdvertise(), Ip6SelectInterface(), Ip6SelectSourceAddress(), and Ip6SetAddress().

```
                                    IN
BOOLEAN Ip6IsSNMulticastAddr  (  EFI_IPv6_ADDRESS   Ip6   )
                                    *
```

Check whether the incoming Ipv6 address is a solicited-node multicast address.

Parameters:

[in]	*Ip6*	Ip6 address, in network order.

Return values:

TRUE	Yes, solicited-node multicast address
FALSE	No

References Ip6CreateSNMulticastAddr().

Referenced by Ip6ProcessNeighborSolicit(), and Ip6SendNeighborSolicit().

```
BOOLEAN Ip6IsValidLinkAddress ( IN IP6_SERVICE *          IpSb,
                                IN EFI_MAC_ADDRESS *      LinkAddress
                              )
```

Check whether the incoming MAC address is valid.

Parameters:

[in]	*IpSb*	Points to a IP6 service binding instance.
[in]	*LinkAddress*	The MAC address.

Return values:

TRUE	Yes, it is valid.
FALSE	No, it is not valid.

Referenced by EfiIp6Neighbors().

```
EFI_IP6_HEADER* Ip6NtohHead ( IN OUT EFI_IP6_HEADER * Head )
```

Convert the multibyte field in IP header's byter order. In spite of its name, it can also be used to convert from host to network byte order.

Parameters:

[in,out]	*Head*	The IP head to convert.

Returns:

Point to the converted IP head.

Referenced by Ip6IpSecProcessPacket(), Ip6PreProcessPacket(), Ip6Reassemble(), and Ip6WrapRxData().

```
EFI_STATUS Ip6RemoveAddr ( IN IP6_SERVICE *          IpSb,
                           IN OUT LIST_ENTRY *       AddressList,
                           IN OUT UINT32 *           AddressCount,
                           IN EFI_IPv6_ADDRESS *     Prefix,
                           IN UINT8                  PrefixLength
                         )
```

Remove the IPv6 address from the address list node points to IP6_ADDRESS_INFO.

This function removes the matching IPv6 addresses from the address list and adjusts the address count of the address list. If IpSb is not NULL, this function calls Ip6LeaveGroup to see whether it should call Mnp->Groups() to remove the its solicited-node multicast MAC address from the filter list and sends out a Multicast Listener Done. If Prefix is NULL, all address in the address list will be removed. If Prefix is not NULL, the address that matching the Prefix with PrefixLength in the address list will be removed.

Parameters:

[in]	*IpSb*	NULL or points to IP6 service binding instance.
[in,out]	*AddressList*	address list array
[in,out]	*AddressCount*	the count of addresses in address list array
[in]	*Prefix*	NULL or an IPv6 address prefix
[in]	*PrefixLength*	the length of Prefix

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_FOUND	The address matching the Prefix with PrefixLength cannot be found in address list.
EFI_INVALID_PARAMETER	Any input parameter is invalid.

Remove the IPv6 address from the address list node points to IP6_ADDRESS_INFO.

This function removes the matching IPv6 addresses from the address list and adjusts the address count of the address list. If IpSb is not NULL, this function calls Ip6LeaveGroup to see whether it should call Mnp->Groups() to remove the its solicited-node multicast MAC address from the filter list and sends out a Multicast Listener Done. If Prefix is NULL, all address in the address list will be removed. If Prefix is not NULL, the address that matching the Prefix with PrefixLength in the address list will be removed.

Parameters:

`[in]`	*IpSb*	NULL or points to IP6 service binding instance.
`[in,out]`	*AddressList*	Address list array.
`[in,out]`	*AddressCount*	The count of addresses in address list array.
`[in]`	*Prefix*	NULL or an IPv6 address prefix.
`[in]`	*PrefixLength*	The length of Prefix.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_FOUND	The address matching the Prefix with PrefixLength cannot be found in the address list.
EFI_INVALID_PARAMETER	Any input parameter is invalid.

References _IP6_ADDRESS_INFO::Address, IP6_ADDR_INFO_SIGNATURE, IP6_SERVICE_SIGNATURE, Ip6CreateSNMulticastAddr(), Ip6DestroyInstanceByAddress(), Ip6LeaveGroup(), and _IP6_ADDRESS_INFO::PrefixLength.

Referenced by Ip6CleanInterface(), Ip6ConfigOnDhcp6Event(), Ip6ConfigOnPolicyChanged(), Ip6ConfigSetMaunualAddress(), Ip6DestroyPrefixListEntry(), and Ip6SelectSourceAddress().

```
EFI_STATUS Ip6SetToAllNodeMulticast ( IN BOOLEAN                Router,
                                      IN UINT8                  Scope,
                                      OUT EFI_IPv6_ADDRESS *    Ip6Addr
                                    )
```

Generate the multicast addresses identify the group of all IPv6 nodes or IPv6 routers defined in RFC4291.

All Nodes Addresses: FF01::1, FF02::1. All Router Addresses: FF01::2, FF02::2, FF05::2.

Parameters:

`[in]`	*Router*	If TRUE, generate all routers addresses, else generate all node addresses.
`[in]`	*Scope*	interface-local(1), link-local(2), or site-local(5)
`[out]`	*Ip6Addr*	The generated multicast address.

Return values:

EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_SUCCESS	The address is generated.

References IP6_SITE_LOCAL_SCOPE.

Referenced by Ip6CleanService(), Ip6InitMld(), Ip6OnDADFinished(), Ip6ProcessMldQuery(), Ip6ProcessNeighborSolicit(), Ip6SendMldDone(), and Ip6SendRouterSolicit().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Ip6Dxe/Ip6Config.vfr File Reference

Defines

"memItemRight" valign="bottom">EFI_NETWORK_DEVICE_CLASS 0x04

Variables

```
formset guid = IP6_CONFIG_NVDATA_GU
formset title =
        STRING_TOKEN(STR_IP6_CONFI
formset help =
        STRING_TOKEN(STR_IP6_CONFI
formset varstore IP6_CONFIG_IFR_NVDATA
formset varstore name = IP6_CONFIG_IFR_NVDATA
form formid = FORMID_HEAD_FORM
goto FORMID_MAIN_FORM
goto prompt = STRING_TOKEN
        (STR_GET_CURRENT_SETTING)
goto flags = INTERACTIVE
goto key = KEY_GET_CURRENT_SETT
        endform
text text =
        STRING_TOKEN(STR_IP6_INTER
label HOST_ADDRESS_LABEL
label LABEL_END
label ROUTE_TABLE_LABEL
label GATEWAY_ADDRESS_LABEL
label DNS_ADDRESS_LABEL
string varid = IP6_CONFIG_IFR_NVDATA
string minsize = INTERFACE_ID_STR_MI
string maxsize = INTERFACE_ID_STR_M
string endstring
numeric minimum = 0
numeric maximum = DAD_MAX_TRANSMI
numeric step = 0
numeric endnumeric
oneof option value = IP6_POLICY_AUTO
        endoneof
suppressif ideqval
IP6_CONFIG_IFR_NVDATA Policy = = IP6_POLICY_AUTO
goto FORMID_MANUAL_CONFIG_FOR
        endif
        endformset
```

Detailed Description

VFR file used by the IP6 configuration component.

Copyright (c) 2010 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Variable Documentation

label DNS_ADDRESS_LABEL

endform

endformset

endif

numeric endnumeric

endoneof

string endstring

goto flags = INTERACTIVE

form formid = FORMID_HEAD_FORM

goto FORMID_MAIN_FORM

goto FORMID_MANUAL_CONFIG_FORM

label GATEWAY_ADDRESS_LABEL

formset varstore guid = IP6_CONFIG_NVDATA_GUID

goto help = STRING_TOKEN(STR_IP6_CONFIG_FORM_HELP)

label HOST_ADDRESS_LABEL

formset varstore IP6_CONFIG_IFR_NVDATA

text key = KEY_GET_CURRENT_SETTING

label LABEL_END

numeric maximum = DAD_MAX_TRANSMIT_COUNT

string maxsize = INTERFACE_ID_STR_MAX_SIZE

numeric minimum = 0

string minsize = INTERFACE_ID_STR_MIN_SIZE

formset varstore name = IP6_CONFIG_IFR_NVDATA

suppressif ideqval IP6_CONFIG_IFR_NVDATA Policy = = IP6_POLICY_AUTO

Referenced by HttpBootSetIpPolicy(), IfConfig6ClearInterfaceInfo(), IfConfig6SetInterfaceInfo(), Ip6ConvertConfigNvDataToIfrNvData(), PxeBcRegisterIp6Address(), PxeBcSetIp4Policy(), and PxeBcSetIp6Policy().

goto prompt = STRING_TOKEN (STR_GET_CURRENT_SETTING)

label ROUTE_TABLE_LABEL

numeric step = 0

text text = STRING_TOKEN(STR_IP6_INTERFACE_NAME)

form title = STRING_TOKEN(STR_IP6_CONFIG_FORM_TITLE)

option value = IP6_POLICY_AUTO

string varid = IP6_CONFIG_IFR_NVDATA.InterfaceId

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Ip6Dxe/Ip6ConfigImpl.c File Reference

Functions

VOID EFIAPI Ip6ConfigOnDhcp6SbInstalled (IN EFI_EVENT Event, IN VOID *Context)
VOID Ip6ConfigOnPolicyChanged (IN IP6_SERVICE *IpSb, IN EFI_IP6_CONFIG_POLICY NewPolicy)
EFI_STATUS Ip6ConfigStartStatefulAutoConfig (IN IP6_CONFIG_INSTANCE *Instance, IN BOOLEAN OtherInfoOnly)
EFI_STATUS EFIAPI Ip6ConfigSignalEvent (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Arg)
EFI_STATUS Ip6ConfigReadConfigData (IN CHAR16 *VarName, IN OUT IP6_CONFIG_INSTANCE *Instance)
EFI_STATUS Ip6ConfigWriteConfigData (IN CHAR16 *VarName, IN IP6_CONFIG_INSTANCE *Instance)
EFI_STATUS Ip6ConfigGetIfInfo (IN IP6_CONFIG_INSTANCE *Instance, IN OUT UINTN *DataSize, IN VOID *Data)
EFI_STATUS Ip6ConfigSetAltIfId (IN IP6_CONFIG_INSTANCE *Instance, IN UINTN DataSize, IN VOID *Data)
EFI_STATUS Ip6ConfigSetPolicy (IN IP6_CONFIG_INSTANCE *Instance, IN UINTN DataSize, IN VOID *Data)
EFI_STATUS Ip6ConfigSetDadXmits (IN IP6_CONFIG_INSTANCE *Instance, IN UINTN DataSize, IN VOID *Data)
VOID Ip6ManualAddrDadCallback (IN BOOLEAN IsDadPassed, IN EFI_IPv6_ADDRESS *TargetAddress, IN VOID *Context)
EFI_STATUS Ip6ConfigSetMaunualAddress (IN IP6_CONFIG_INSTANCE *Instance, IN UINTN DataSize, IN VOID *Data)
EFI_STATUS Ip6ConfigSetGateway (IN IP6_CONFIG_INSTANCE *Instance, IN UINTN DataSize, IN VOID *Data)
EFI_STATUS Ip6ConfigSetDnsServer (IN IP6_CONFIG_INSTANCE *Instance, IN UINTN DataSize, IN VOID *Data)
VOID Ip6ConfigInitIfInfo (IN IP6_SERVICE *IpSb, OUT EFI_IP6_CONFIG_INTERFACE_INFO *IfInfo)
EFI_STATUS Ip6ConfigParseDhcpReply (IN EFI_DHCP6_PROTOCOL *Dhcp6, IN OUT IP6_CONFIG_INSTANCE *Instance, IN EFI_DHCP6_PACKET *Reply)
VOID Ip6ConfigSetStatefulAddrCallback (IN BOOLEAN IsDadPassed, IN EFI_IPv6_ADDRESS *TargetAddress, IN VOID *Context)
VOID EFIAPI Ip6ConfigOnDhcp6Event (IN EFI_EVENT Event, IN VOID *Context)
EFI_STATUS EFIAPI Ip6ConfigOnDhcp6Reply (IN EFI_DHCP6_PROTOCOL *This, IN VOID *Context, IN EFI_DHCP6_PACKET *Packet)
EFI_STATUS EFIAPI EfiIp6ConfigSetData (IN EFI_IP6_CONFIG_PROTOCOL *This, IN EFI_IP6_CONFIG_DATA_TYPE DataType, IN UINTN DataSize, IN VOID *Data)
EFI_STATUS EFIAPI EfiIp6ConfigGetData (IN EFI_IP6_CONFIG_PROTOCOL *This, IN EFI_IP6_CONFIG_DATA_TYPE DataType, IN OUT UINTN *DataSize, IN VOID *Data)
EFI_STATUS EFIAPI EfiIp6ConfigRegisterDataNotify (IN EFI_IP6_CONFIG_PROTOCOL *This, IN EFI_IP6_CONFIG_DATA_TYPE DataType, IN EFI_EVENT Event)
EFI_STATUS EFIAPI EfiIp6ConfigUnregisterDataNotify (IN EFI_IP6_CONFIG_PROTOCOL *This, IN EFI_IP6_CONFIG_DATA_TYPE DataType, IN EFI_EVENT Event)
EFI_STATUS Ip6ConfigInitInstance (OUT IP6_CONFIG_INSTANCE *Instance)
VOID Ip6ConfigCleanInstance (IN OUT IP6_CONFIG_INSTANCE *Instance)
EFI_STATUS Ip6ConfigDestroyDhcp6 (IN OUT IP6_CONFIG_INSTANCE *Instance)

Variables

LIST_ENTRY mIp6ConfigInstanceList = {&mIp6ConfigInstanceList, &mIp6ConfigInstanceList}

Detailed Description

The implementation of EFI IPv6 Configuration Protocol.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI EfiIp6ConfigGetData ( IN EFI_IP6_CONFIG_PROTOCOL *  This,
                                        IN EFI_IP6_CONFIG_DATA_TYPE   DataType,
                                        IN OUT UINTN *                DataSize,
                                        IN VOID *                     Data
                                      )
```

Get the configuration data for the EFI IPv6 network stack running on the communication device that this EFI IPv6 Configuration Protocol instance manages.

This function returns the configuration data of type DataType for the EFI IPv6 network stack running on the communication device that this EFI IPv6 Configuration Protocol instance manages.

The caller is responsible for allocating the buffer used to return the specified configuration data. The required size will be returned to the caller if the size of the buffer is too small.

EFI_NOT_READY is returned if the specified configuration data is not ready due to an asynchronous configuration process already in progress. The caller can call RegisterDataNotify() to register an event on the specified configuration data. Once the asynchronous configuration process is finished, the event will be signaled, and a subsequent GetData() call will return the specified configuration data.

Parameters:

[in]	*This*	Pointer to the EFI_IP6_CONFIG_PROTOCOL instance.
[in]	*DataType*	The type of data to get.
[in,out]	*DataSize*	On input, in bytes, the size of Data. On output, in bytes, the size of buffer required to store the specified configuration data.
[in]	*Data*	The data buffer in which the configuration data is returned. The type of the data buffer is associated with the DataType. This is an optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_INVALID_PARAMETER	One or more of the followings are TRUE: ◊ This is NULL. ◊ DataSize is NULL. ◊ Data is NULL if *DataSize is not zero.
EFI_BUFFER_TOO_SMALL	The size of Data is too small for the specified configuration data, and the required size is returned in DataSize.
EFI_NOT_READY	The specified configuration data is not ready due to an asynchronous configuration process already in progress.
EFI_NOT_FOUND	The specified configuration data is not found.

References IP6_CONFIG_DATA_ITEM::Data, _IP6_CONFIG_INSTANCE::DataItem, IP6_CONFIG_DATA_ITEM::DataSize, IP6_CONFIG_DATA_ITEM::GetData, IP6_CONFIG_INSTANCE_FROM_PROTOCOL, IP6_CONFIG_DATA::Ptr, and IP6_CONFIG_DATA_ITEM::Status.

Referenced by Ip6ConfigInitInstance().

```
EFI_STATUS EFIAPI
EfiIp6ConfigRegisterDataNotify ( IN EFI_IP6_CONFIG_PROTOCOL *  This,
                                 IN EFI_IP6_CONFIG_DATA_TYPE   DataType,
                                 IN EFI_EVENT                  Event
                                 )
```

Register an event that is signaled whenever a configuration process on the specified configuration data is done.

This function registers an event that is to be signaled whenever a configuration process on the specified configuration data is performed. An event can be registered for a different DataType simultaneously. The caller is responsible for determining which type of configuration data causes the signaling of the event in such an event.

Parameters:

[in]	*This*	Pointer to the EFI_IP6_CONFIG_PROTOCOL instance.
[in]	*DataType*	The type of data to unregister the event for.
[in]	*Event*	The event to register.

Return values:

EFI_SUCCESS	The notification event for the specified configuration data is registered.
EFI_INVALID_PARAMETER	This is NULL or Event is NULL.
EFI_UNSUPPORTED	The configuration data type specified by DataType is not supported.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_ACCESS_DENIED	The Event is already registered for the DataType.

References _IP6_CONFIG_INSTANCE::DataItem, IP6_CONFIG_DATA_ITEM::EventMap, and IP6_CONFIG_INSTANCE_FROM_PROTOCOL.

Referenced by Ip6ConfigInitInstance().

```
EFI_STATUS EFIAPI EfiIp6ConfigSetData ( IN EFI_IP6_CONFIG_PROTOCOL *  This,
```

```
                    IN EFI_IP6_CONFIG_DATA_TYPE   DataType,
                    IN UINTN                      DataSize,
                    IN VOID *                     Data
                  )
```

Set the configuration for the EFI IPv6 network stack running on the communication device this EFI IPv6 Configuration Protocol instance manages.

This function is used to set the configuration data of type DataType for the EFI IPv6 network stack that is running on the communication device that this EFI IPv6 Configuration Protocol instance manages.

DataSize is used to calculate the count of structure instances in the Data for a DataType in which multiple structure instances are allowed.

This function is always non-blocking. When setting some type of configuration data, an asynchronous process is invoked to check the correctness of the data, such as performing Duplicate Address Detection on the manually set local IPv6 addresses. EFI_NOT_READY is returned immediately to indicate that such an asynchronous process is invoked, and the process is not finished yet. The caller wanting to get the result of the asynchronous process is required to call RegisterDataNotify() to register an event on the specified configuration data. Once the event is signaled, the caller can call GetData() to obtain the configuration data and know the result. For other types of configuration data that do not require an asynchronous configuration process, the result of the operation is immediately returned.

Parameters:

`[in]`	*This*	The pointer to the EFI_IP6_CONFIG_PROTOCOL instance.
`[in]`	*DataType*	The type of data to set.
`[in]`	*DataSize*	Size of the buffer pointed to by Data in bytes.
`[in]`	*Data*	The data buffer to set. The type of the data buffer is associated with the DataType.

Return values:

EFI_SUCCESS	The specified configuration data for the EFI IPv6 network stack was set successfully.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: ◊ This is NULL. ◊ Data is NULL. ◊ One or more fields in Data do not match the requirement of the data type indicated by DataType.
EFI_WRITE_PROTECTED	The specified configuration data is read-only or the specified configuration data cannot be set under the current policy.
EFI_ACCESS_DENIED	Another set operation on the specified configuration data is already in process.
EFI_NOT_READY	An asynchronous process was invoked to set the specified configuration data, and the process is not finished yet.
EFI_BAD_BUFFER_SIZE	The DataSize does not match the size of the type indicated by DataType.
EFI_UNSUPPORTED	This DataType is not supported.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_DEVICE_ERROR	An unexpected system error or network error occurred.

References _IP6_CONFIG_INSTANCE::DataItem, IP6_CONFIG_DATA_ITEM::EventMap, IP6_CONFIG_INSTANCE_FROM_PROTOCOL, IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE, IP6_SERVICE_SIGNATURE, Ip6ConfigSignalEvent(), Ip6ConfigWriteConfigData(), _IP6_SERVICE::LinkLocalDadFail, _IP6_SERVICE::MacString, IP6_CONFIG_DATA_ITEM::SetData,

and IP6_CONFIG_DATA_ITEM::Status.

Referenced by Ip6ConfigInitInstance().

EFI_STATUS EFIAPI EfiIp6ConfigUnregisterDataNotify (IN EFI_IP6_CONFIG_PROTOCOL * *This*, IN EFI_IP6_CONFIG_DATA_TYPE *DataType*, IN EFI_EVENT *Event*)

Remove a previously registered event for the specified configuration data.

Parameters:

This	The pointer to the EFI_IP6_CONFIG_PROTOCOL instance.
DataType	The type of data to remove from the previously registered event.
Event	The event to be unregistered.

Return values:

EFI_SUCCESS	The event registered for the specified configuration data was removed.
EFI_INVALID_PARAMETER	This is NULL or Event is NULL.
EFI_NOT_FOUND	The Event has not been registered for the specified DataType.

References _IP6_CONFIG_INSTANCE::DataItem, IP6_CONFIG_DATA_ITEM::EventMap, and IP6_CONFIG_INSTANCE_FROM_PROTOCOL.

Referenced by Ip6ConfigInitInstance().

VOID Ip6ConfigCleanInstance (IN OUT IP6_CONFIG_INSTANCE * *Instance*)

Release an IP6_CONFIG_INSTANCE.

Parameters:

`[in,out]` *Instance* The buffer of IP6_CONFIG_INSTANCE to be freed.

References IP6_CONFIG_DATA_ITEM::Attribute, IP6_CONFIG_DATA_ITEM::Data, DATA_ATTRIB_SET, DATA_ATTRIB_SIZE_FIXED, IP6_CONFIG_DATA_ITEM::DataSize, Ip6ConfigDestroyDhcp6(), Ip6ConfigFormUnload(), and IP6_CONFIG_DATA::Ptr.

Referenced by Ip6CleanService().

EFI_STATUS Ip6ConfigDestroyDhcp6 (IN OUT IP6_CONFIG_INSTANCE * *Instance*)

Destroy the Dhcp6 child in IP6_CONFIG_INSTANCE and release the resources.

Parameters:

`[in,out]` *Instance* The buffer of IP6_CONFIG_INSTANCE to be freed.

Return values:

EFI_SUCCESS	The child was successfully destroyed.
Others	Failed to destroy the child.

References _IP6_SERVICE::Controller, _IP6_SERVICE::Image, and IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE.

Referenced by Ip6ConfigCleanInstance(), Ip6ConfigSetPolicy(), and Ip6DriverBindingStop().

```
EFI_STATUS Ip6ConfigGetIfInfo ( IN IP6_CONFIG_INSTANCE *   Instance,
                                IN OUT UINTN *             DataSize,
                                IN VOID *                  Data
                              )
```

The work function for EfiIp6ConfigGetData() to get the interface information of the communication device this IP6Config instance manages.

Parameters:

[in]	*Instance*	Pointer to the IP6 config instance data.
[in,out]	*DataSize*	On input, in bytes, the size of Data. On output, in bytes, the size of buffer required to store the specified configuration data.
[in]	*Data*	The data buffer in which the configuration data is returned. Ignored if DataSize is ZERO.

Return values:

EFI_BUFFER_TOO_SMALL	The size of Data is too small for the specified configuration data, and the required size is returned in DataSize.
EFI_SUCCESS	The specified configuration data was obtained.

References IP6_CONFIG_DATA_ITEM::Data, IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE, Ip6BuildEfiAddressList(), Ip6BuildEfiRouteTable(), IP6_CONFIG_DATA::Ptr, and _IP6_SERVICE::RouteTable.

Referenced by Ip6ConfigInitInstance().

```
VOID Ip6ConfigInitIfInfo ( IN IP6_SERVICE *                        IpSb,
                           OUT EFI_IP6_CONFIG_INTERFACE_INFO *     IfInfo
                         )
```

Generate the operational state of the interface this IP6 config instance manages and output in EFI_IP6_CONFIG_INTERFACE_INFO.

Parameters:

[in]	*IpSb*	The pointer to the IP6 service binding instance.
[out]	*IfInfo*	The pointer to the IP6 configuration interface information structure.

Referenced by Ip6ConfigInitInstance().

```
EFI_STATUS Ip6ConfigInitInstance ( OUT IP6_CONFIG_INSTANCE *   Instance   )
```

Initialize an IP6_CONFIG_INSTANCE.

Parameters:

[out]	*Instance*	The buffer of IP6_CONFIG_INSTANCE to be initialized.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate resources to complete the operation.
EFI_SUCCESS	The IP6_CONFIG_INSTANCE initialized successfully.

References IP6_CONFIG_DATA_ITEM::Attribute, IP6_CONFIG_DATA_ITEM::Data, DATA_ATTRIB_SIZE_FIXED, DATA_ATTRIB_VOLATILE, IP6_CONFIG_DATA_ITEM::DataSize, EfiIp6ConfigGetData(), EfiIp6ConfigRegisterDataNotify(), EfiIp6ConfigSetData(), EfiIp6ConfigUnregisterDataNotify(), IP6_CONFIG_DATA_ITEM::GetData, _IP6_CONFIG_INSTANCE::IfIndex, IP6_CONFIG_DEFAULT_DAD_XMITS, IP6_CONFIG_INSTANCE_SIGNATURE, IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE, Ip6ConfigFormInit(), Ip6ConfigGetIfInfo(), Ip6ConfigInitIfInfo(), Ip6ConfigOnDhcp6Event(), Ip6ConfigReadConfigData(), Ip6ConfigSetAltIfId(), Ip6ConfigSetDadXmits(), Ip6ConfigSetDnsServer(), Ip6ConfigSetGateway(), Ip6ConfigSetMaunualAddress(), Ip6ConfigSetPolicy(), Ip6ConfigWriteConfigData(), _IP6_ADDRESS_INFO::Link, _IP6_SERVICE::MacString, mIp6ConfigInstanceList, IP6_CONFIG_DATA::Ptr, SET_DATA_ATTRIB, IP6_CONFIG_DATA_ITEM::SetData, _IP6_SERVICE::SnpMode, and IP6_CONFIG_DATA_ITEM::Status.

Referenced by Ip6CreateService().

```
VOID EFIAPI Ip6ConfigOnDhcp6Event ( IN EFI_EVENT   Event,
                                    IN VOID *      Context
                                  )
```

The event handle routine when DHCPv6 process is finished or is updated.

Parameters:

[in]	*Event*	Not used.
[in]	*Context*	The pointer to the IP6 configuration instance data.

References _IP6_ADDRESS_INFO::Address, _IP6_INTERFACE::AddressCount, _IP6_INTERFACE::AddressList, _IP6_CONFIG_INSTANCE::DeclineAddress, _IP6_CONFIG_INSTANCE::DeclineAddressCount, _IP6_SERVICE::DefaultInterface, _IP6_CONFIG_INSTANCE::Dhcp6, _IP6_CONFIG_INSTANCE::FailedIaAddressCount, IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE, Ip6ConfigParseDhcpReply(), Ip6ConfigSetStatefulAddrCallback(), Ip6IsOneOfSetAddress(), Ip6RemoveAddr(), Ip6SetAddress(), _IP6_CONFIG_INSTANCE::OtherInfoOnly, _IP6_CONFIG_INSTANCE::Policy, and _IP6_ADDRESS_INFO::PrefixLength.

Referenced by Ip6ConfigInitInstance().

```
EFI_STATUS EFIAPI Ip6ConfigOnDhcp6Reply ( IN EFI_DHCP6_PROTOCOL *   This,
                                          IN VOID *                 Context,
                                          IN EFI_DHCP6_PACKET *     Packet
                                        )
```

The event process routine when the DHCPv6 server is answered with a reply packet for an information request.

Parameters:

[in]	*This*	Points to the EFI_DHCP6_PROTOCOL.
[in]	*Context*	The pointer to the IP6 configuration instance data.
[in]	*Packet*	The DHCPv6 reply packet.

Return values:

EFI_SUCCESS	The DNS server address was retrieved from the reply packet.
EFI_NOT_READY	The reply packet does not contain the DNS server option, or the DNS server address is not valid.

References Ip6ConfigParseDhcpReply().

Referenced by Ip6ConfigStartStatefulAutoConfig(), and Ip6OnDADFinished().

```
VOID EFIAPI Ip6ConfigOnDhcp6SbInstalled ( IN EFI_EVENT  Event,
                                          IN VOID *     Context
                                        )
```

The event process routine when the DHCPv6 service binding protocol is installed in the system.

Parameters:

[in]	*Event*	Not used.
[in]	*Context*	Pointer to the IP6 config instance data.

The event process routine when the DHCPv6 service binding protocol is installed in the system.

Parameters:

[in]	*Event*	Not used.
[in]	*Context*	The pointer to the IP6 config instance data.

References _IP6_CONFIG_INSTANCE::Dhcp6Handle, Ip6ConfigStartStatefulAutoConfig(), _IP6_CONFIG_INSTANCE::OtherInfoOnly, and _IP6_CONFIG_INSTANCE::Policy.

Referenced by Ip6ConfigStartStatefulAutoConfig().

```
VOID Ip6ConfigOnPolicyChanged ( IN IP6_SERVICE *              IpSb,
                                IN EFI_IP6_CONFIG_POLICY      NewPolicy
                              )
```

Update the current policy to NewPolicy. During the transition period, the default router list, on-link prefix list, autonomous prefix list and address list in all interfaces will be released.

Parameters:

[in]	*IpSb*	The IP6 service binding instance.
[in]	*NewPolicy*	The new policy to be updated to.

References _IP6_ADDRESS_INFO::Address, _IP6_DAD_ENTRY::AddressInfo, _IP6_DELAY_JOIN_LIST::AddressInfo, _IP6_INTERFACE::DelayJoinList, _IP6_INTERFACE::DupAddrDetectList, IP6_DAD_ENTRY_SIGNATURE, IP6_GET_TICKS, IP6_INFINIT_LIFETIME, IP6_INTERFACE_SIGNATURE, IP6_LINK_LOCAL_PREFIX_LENGTH, IP6_MAX_RTR_SOLICITATIONS, IP6_ONE_SECOND_IN_MS, Ip6CleanDefaultRouterList(), Ip6CleanPrefixListTable(), Ip6CreatePrefixListEntry(), Ip6OnDADFinished(), Ip6RemoveAddr(), _IP6_DELAY_JOIN_LIST::Link, and _IP6_ADDRESS_INFO::Link.

Referenced by Ip6ConfigSetPolicy().

```
EFI_STATUS Ip6ConfigParseDhcpReply ( IN EFI_DHCP6_PROTOCOL *        Dhcp6,
```

```
                                    IN OUT IP6_CONFIG_INSTANCE *   Instance,
                                    IN EFI_DHCP6_PACKET *          Reply
                                  )
```

Parse DHCPv6 reply packet to get the DNS server list. It is the work function for Ip6ConfigOnDhcp6Reply and Ip6ConfigOnDhcp6Event.

Parameters:

[in]	*Dhcp6*	The pointer to the EFI_DHCP6_PROTOCOL instance.
[in,out]	*Instance*	The pointer to the IP6 configuration instance data.
[in]	*Reply*	The pointer to the DHCPv6 reply packet.

Return values:

EFI_SUCCESS	The DNS server address was retrieved from the reply packet.
EFI_NOT_READY	The reply packet does not contain the DNS server option, or the DNS server address is not valid.

References IP6_CONFIG_DATA_ITEM::Data, IP6_CONFIG_DATA_ITEM::DataSize, IP6_CONFIG_DATA_ITEM::EventMap, IP6_CONFIG_DHCP6_OPTION_DNS_SERVERS, Ip6ConfigSignalEvent(), IP6_CONFIG_DATA::Ptr, and IP6_CONFIG_DATA_ITEM::Status.

Referenced by Ip6ConfigOnDhcp6Event(), and Ip6ConfigOnDhcp6Reply().

```
EFI_STATUS Ip6ConfigReadConfigData  (  IN CHAR16 *                  VarName,
                                       IN OUT
                                       IP6_CONFIG_INSTANCE *        Instance
                                    )
```

Read the configuration data from variable storage according to the VarName and gEfiIp6ConfigProtocolGuid. It checks the integrity of variable data. If the data is corrupted, it clears the variable data to ZERO. Othewise, it outputs the configuration data to IP6_CONFIG_INSTANCE.

Parameters:

[in]	*VarName*	The pointer to the variable name
[in,out]	*Instance*	The pointer to the IP6 config instance data.

Return values:

EFI_NOT_FOUND	The variable can not be found or already corrupted.
EFI_OUT_OF_RESOURCES	Fail to allocate resource to complete the operation.
EFI_SUCCESS	The configuration data was retrieved successfully.

References IP6_CONFIG_DATA_ITEM::Attribute, IP6_CONFIG_DATA_ITEM::Data, DATA_ATTRIB_SET, DATA_ATTRIB_SIZE_FIXED, IP6_CONFIG_DATA_RECORD::DataSize, IP6_CONFIG_DATA_ITEM::DataSize, IP6_CONFIG_DATA_RECORD::DataType, IP6_CONFIG_VARIABLE_ATTRIBUTE, IP6_CONFIG_DATA_RECORD::Offset, IP6_CONFIG_DATA::Ptr, and IP6_CONFIG_DATA_ITEM::Status.

Referenced by Ip6ConfigInitInstance().

```
EFI_STATUS Ip6ConfigSetAltIfId  (  IN IP6_CONFIG_INSTANCE *   Instance,
                                   IN UINTN                   DataSize,
                                   IN VOID *                  Data
```

)

The work function for EfiIp6ConfigSetData() to set the alternative inteface ID for the communication device managed by this IP6Config instance, if the link local IPv6 addresses generated from the interface ID based on the default source the EFI IPv6 Protocol uses is a duplicate address.

Parameters:

[in]	*Instance*	Pointer to the IP6 configuration instance data.
[in]	*DataSize*	Size of the buffer pointed to by Data in bytes.
[in]	*Data*	The data buffer to set.

Return values:

EFI_BAD_BUFFER_SIZE	The DataSize does not match the size of the type, 8 bytes.
EFI_SUCCESS	The specified configuration data for the EFI IPv6 network stack was set.

References IP6_CONFIG_DATA::AltIfId, IP6_CONFIG_DATA_ITEM::Data, and IP6_CONFIG_DATA_ITEM::Status.

Referenced by Ip6ConfigInitInstance().

```
EFI_STATUS Ip6ConfigSetDadXmits ( IN IP6_CONFIG_INSTANCE *  Instance,
                                  IN UINTN                  DataSize,
                                  IN VOID *                 Data
                                )
```

The work function for EfiIp6ConfigSetData() to set the number of consecutive Neighbor Solicitation messages sent while performing Duplicate Address Detection on a tentative address. A value of ZERO indicates that Duplicate Address Detection will not be performed on a tentative address.

Parameters:

[in]	*Instance*	The Instance Pointer to the IP6 config instance data.
[in]	*DataSize*	Size of the buffer pointed to by Data in bytes.
[in]	*Data*	The data buffer to set.

Return values:

EFI_BAD_BUFFER_SIZE	The DataSize does not match the size of the type.
EFI_ABORTED	The new transmit count equals the current configuration.
EFI_SUCCESS	The specified configuration data for the EFI IPv6 network stack was set.

Referenced by Ip6ConfigInitInstance().

```
EFI_STATUS Ip6ConfigSetDnsServer ( IN IP6_CONFIG_INSTANCE *  Instance,
                                   IN UINTN                  DataSize,
                                   IN VOID *                 Data
                                 )
```

The work function for EfiIp6ConfigSetData() to set the DNS server list for the EFI IPv6 network stack running on the communication device that this EFI IPv6 Configuration Protocol manages. It is not configurable when the policy is Ip6ConfigPolicyAutomatic. The DNS server addresses must be unicast IPv6 addresses.

Parameters:

[in]	*Instance*	The pointer to the IP6 config instance data.

[in] *DataSize* The size of the buffer pointed to by Data in bytes.
[in] *Data* The data buffer to set, points to an array of EFI_IPv6_ADDRESS instances.

Return values:

EFI_BAD_BUFFER_SIZE	The DataSize does not match the size of the type.
EFI_WRITE_PROTECTED	The specified configuration data cannot be set under the current policy.
EFI_INVALID_PARAMETER	One or more fields in Data is invalid.
EFI_OUT_OF_RESOURCES	Failed to allocate resources to complete the operation.
EFI_ABORTED	The DNS server addresses to be set equal the current configuration.
EFI_SUCCESS	The specified configuration data for the EFI IPv6 network stack was set.

References IP6_CONFIG_DATA_ITEM::Data, IP6_CONFIG_DATA_ITEM::DataSize, IP6_CONFIG_DATA::DnsServers, IP6_CONFIG_DATA::Ptr, and IP6_CONFIG_DATA_ITEM::Status.

Referenced by Ip6ConfigInitInstance().

```
EFI_STATUS Ip6ConfigSetGateway ( IN IP6_CONFIG_INSTANCE *  Instance,
                                 IN UINTN                  DataSize,
                                 IN VOID *                 Data
                               )
```

The work function for EfiIp6ConfigSetData() to set the gateway addresses manually for the EFI IPv6 network stack that is running on the communication device that this EFI IPv6 Configuration Protocol manages. It is not configurable when the policy is Ip6ConfigPolicyAutomatic. The gateway addresses must be unicast IPv6 addresses.

Parameters:

[in] *Instance* The pointer to the IP6 config instance data.
[in] *DataSize* The size of the buffer pointed to by Data in bytes.
[in] *Data* The data buffer to set. This points to an array of EFI_IPv6_ADDRESS instances.

Return values:

EFI_BAD_BUFFER_SIZE	The DataSize does not match the size of the type.
EFI_WRITE_PROTECTED	The specified configuration data cannot be set under the current policy.
EFI_INVALID_PARAMETER	One or more fields in Data is invalid.
EFI_OUT_OF_RESOURCES	Failed to allocate resource to complete the operation.
EFI_ABORTED	The manual gateway addresses to be set equal the current configuration.
EFI_SUCCESS	The specified configuration data for the EFI IPv6 network stack was set.

References IP6_CONFIG_DATA_ITEM::Data, IP6_CONFIG_DATA_ITEM::DataSize, IP6_CONFIG_DATA::Gateway, IP6_INF_ROUTER_LIFETIME, IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE, Ip6CreateDefaultRouter(), Ip6DestroyDefaultRouter(), Ip6FindDefaultRouter(), IP6_CONFIG_DATA::Ptr, and IP6_CONFIG_DATA_ITEM::Status.

Referenced by Ip6ConfigInitInstance().

```
EFI_STATUS Ip6ConfigSetMaunualAddress ( IN IP6_CONFIG_INSTANCE *  Instance,
                                        IN UINTN                  DataSize,
                                        IN VOID *                 Data
                                      )
```

The work function for EfiIp6ConfigSetData() to set the station addresses manually for the EFI IPv6 network stack. It is only configurable when the policy is Ip6ConfigPolicyManual.

Parameters:

[in]	*Instance*	Pointer to the IP6 configuration instance data.
[in]	*DataSize*	Size of the buffer pointed to by Data in bytes.
[in]	*Data*	The data buffer to set.

Return values:

EFI_BAD_BUFFER_SIZE	The DataSize does not match the size of the type.
EFI_WRITE_PROTECTED	The specified configuration data cannot be set under the current policy.
EFI_INVALID_PARAMETER	One or more fields in Data is invalid.
EFI_OUT_OF_RESOURCES	Fail to allocate resource to complete the operation.
EFI_NOT_READY	An asynchrous process is invoked to set the specified configuration data, and the process is not finished.
EFI_ABORTED	The manual addresses to be set equal current configuration.
EFI_SUCCESS	The specified configuration data for the EFI IPv6 network stack was set.

References _IP6_ADDRESS_INFO::Address, _IP6_INTERFACE::AddressCount, _IP6_INTERFACE::AddressList, IP6_CONFIG_DATA_ITEM::Data, IP6_CONFIG_DATA_ITEM::DataSize, _IP6_SERVICE::DefaultInterface, _IP6_SERVICE::Interfaces, IP6_ADDR_INFO_SIGNATURE, IP6_INFINIT_LIFETIME, IP6_INTERFACE_SIGNATURE, IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE, Ip6CreatePrefixListEntry(), Ip6DestroyPrefixListEntry(), Ip6FindPrefixListEntry(), Ip6IsOneOfSetAddress(), Ip6ManualAddrDadCallback(), Ip6RemoveAddr(), Ip6SetAddress(), _IP6_ADDRESS_INFO::IsAnycast, _IP6_ADDRESS_INFO::Link, _IP6_ADDRESS_INFO::PrefixLength, IP6_CONFIG_DATA::Ptr, and IP6_CONFIG_DATA_ITEM::Status.

Referenced by Ip6ConfigInitInstance().

```
EFI_STATUS Ip6ConfigSetPolicy ( IN IP6_CONFIG_INSTANCE *  Instance,
                                IN UINTN                  DataSize,
                                IN VOID *                 Data
                              )
```

The work function for EfiIp6ConfigSetData() to set the general configuration policy for the EFI IPv6 network stack that is running on the communication device managed by this IP6Config instance. The policy will affect other configuration settings.

Parameters:

[in]	*Instance*	Pointer to the IP6 config instance data.
[in]	*DataSize*	Size of the buffer pointed to by Data in bytes.
[in]	*Data*	The data buffer to set.

Return values:

EFI_INVALID_PARAMETER	The to be set policy is invalid.

EFI_BAD_BUFFER_SIZE	The DataSize does not match the size of the type.
EFI_ABORTED	The new policy equals the current policy.
EFI_SUCCESS	The specified configuration data for the EFI IPv6 network stack was set.

References IP6_CONFIG_DATA_ITEM::Data, IP6_CONFIG_DATA_ITEM::DataSize, IP6_CONFIG_DATA_ITEM::EventMap, IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE, Ip6ConfigDestroyDhcp6(), Ip6ConfigOnPolicyChanged(), Ip6ConfigSignalEvent(), IP6_CONFIG_DATA::Ptr, and IP6_CONFIG_DATA_ITEM::Status.

Referenced by Ip6ConfigInitInstance().

```
VOID Ip6ConfigSetStatefulAddrCallback ( IN BOOLEAN              IsDadPassed,
                                        IN EFI_IPv6_ADDRESS *   TargetAddress,
                                        IN VOID *               Context
                                      )
```

The callback function for Ip6SetAddr. The prototype is defined as IP6_DAD_CALLBACK. It is called after Duplicate Address Detection is performed on the tentative address by DHCPv6 in Ip6ConfigOnDhcp6Event().

Parameters:

[in]	*IsDadPassed*	If TRUE, Duplicate Address Detection passes.
[in]	*TargetAddress*	The tentative IPv6 address to be checked.
[in]	*Context*	Pointer to the IP6 configuration instance data.

References _IP6_CONFIG_INSTANCE::DeclineAddress, _IP6_CONFIG_INSTANCE::DeclineAddressCount, _IP6_CONFIG_INSTANCE::Dhcp6, _IP6_CONFIG_INSTANCE::FailedIaAddressCount, and IP6_CONFIG_INSTANCE_SIGNATURE.

Referenced by Ip6ConfigOnDhcp6Event().

```
EFI_STATUS EFIAPI Ip6ConfigSignalEvent ( IN NET_MAP *          Map,
                                         IN NET_MAP_ITEM *     Item,
                                         IN VOID *             Arg
                                       )
```

Signal the registered event. It is the callback routine for NetMapIterate.

Parameters:

[in]	*Map*	Points to the list of registered event.
[in]	*Item*	The registered event.
[in]	*Arg*	Not used.

Referenced by EfiIp6ConfigSetData(), Ip6ConfigParseDhcpReply(), Ip6ConfigSetPolicy(), and Ip6ManualAddrDadCallback().

```
EFI_STATUS Ip6ConfigStartStatefulAutoConfig ( IN IP6_CONFIG_INSTANCE *  Instance,
                                              IN BOOLEAN                OtherInfoOnly
                                            )
```

The work function to trigger the DHCPv6 process to perform a stateful autoconfiguration.

Parameters:

[in]	*Instance*	Pointer to the IP6 config instance data.
[in]	*OtherInfoOnly*	If FALSE, get stateful address and other information via DHCPv6. Otherwise, only get the other information.

Return values:

EFI_SUCCESS	The operation finished successfully.
EFI_UNSUPPORTED	The DHCP6 driver is not available.

References _IP6_SERVICE::Controller, _IP6_SERVICE::Dhcp6NeedInfoRequest, _IP6_SERVICE::Dhcp6NeedStart, _IP6_SERVICE::Image, IP6_CONFIG_DHCP6_OPTION_DNS_SERVERS, IP6_CONFIG_DHCP6_OPTION_ORO, IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE, Ip6ConfigOnDhcp6Reply(), Ip6ConfigOnDhcp6SbInstalled(), and _IP6_SERVICE::LinkLocalOk.

Referenced by Ip6ConfigOnDhcp6SbInstalled(), and Ip6ProcessRouterAdvertise().

```
EFI_STATUS Ip6ConfigWriteConfigData ( IN CHAR16 *                VarName,
                                      IN IP6_CONFIG_INSTANCE *   Instance
                                    )
```

Write the configuration data from IP6_CONFIG_INSTANCE to variable storage.

Parameters:

[in]	*VarName*	The pointer to the variable name.
[in]	*Instance*	The pointer to the IP6 configuration instance data.

Return values:

EFI_OUT_OF_RESOURCES	Fail to allocate resource to complete the operation.
EFI_SUCCESS	The configuration data is written successfully.

References IP6_CONFIG_DATA_ITEM::Attribute, IP6_CONFIG_VARIABLE::Checksum, IP6_CONFIG_DATA_ITEM::Data, DATA_ATTRIB_SET, DATA_ATTRIB_VOLATILE, IP6_CONFIG_VARIABLE::DataRecord, IP6_CONFIG_VARIABLE::DataRecordCount, IP6_CONFIG_DATA_RECORD::DataSize, IP6_CONFIG_DATA_ITEM::DataSize, IP6_CONFIG_DATA_RECORD::DataType, IP6_CONFIG_VARIABLE::IaId, IP6_CONFIG_VARIABLE_ATTRIBUTE, IP6_CONFIG_DATA_RECORD::Offset, IP6_CONFIG_DATA::Ptr, and IP6_CONFIG_DATA_ITEM::Status.

Referenced by EfiIp6ConfigSetData(), Ip6ConfigInitInstance(), and Ip6ManualAddrDadCallback().

```
VOID Ip6ManualAddrDadCallback ( IN BOOLEAN            IsDadPassed,
                                IN EFI_IPv6_ADDRESS * TargetAddress,
                                IN VOID *             Context
                              )
```

The callback function for Ip6SetAddr. The prototype is defined as IP6_DAD_CALLBACK. It is called after Duplicate Address Detection is performed for the manual address set by Ip6ConfigSetMaunualAddress.

Parameters:

[in]	*IsDadPassed*	If TRUE, Duplicate Address Detection passed.
[in]	*TargetAddress*	The tentative IPv6 address to be checked.

[in] *Context* Pointer to the IP6 configuration instance data.

References _IP6_CONFIG_INSTANCE::DadFailedMap, _IP6_CONFIG_INSTANCE::DadPassedMap, IP6_CONFIG_DATA_ITEM::Data, _IP6_CONFIG_INSTANCE::DataItem, IP6_CONFIG_DATA_ITEM::DataSize, IP6_CONFIG_DATA_ITEM::EventMap, IP6_CONFIG_INSTANCE_SIGNATURE, IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE, Ip6ConfigSignalEvent(), Ip6ConfigWriteConfigData(), _IP6_SERVICE::MacString, IP6_CONFIG_DATA::ManualAddress, IP6_CONFIG_DATA::Ptr, and IP6_CONFIG_DATA_ITEM::Status.

Referenced by Ip6ConfigSetMaunualAddress().

Variable Documentation

LIST_ENTRY mIp6ConfigInstanceList = {&mIp6ConfigInstanceList, &mIp6ConfigInstanceList}

Referenced by Ip6ConfigInitInstance().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Ip6Dxe/Ip6ConfigImpl.h File Reference

Data Structures

union IP6
struct IP6
struct IP6
struct IP6
struct IP6
struct IP6
struct _IP
struct _IP

Defines

"memItemRight" valign="bottom">IP6_CONFIG_INSTANCE_SIGNATURE SIGNATURE_32 ('I', 'P', '6', 'C')
#define IP6
'C',
#define IP6
| EF
#define IP6
#define IP6
#define IP6
#define DA
#define DA
#define DA
#define SET
#define IP6
#define IP6
#define IP6
#define IP6

Typedefs

typedef struct _IP6_CONFIG_INSTANCE IP6
typedef EFI_STATUS(* IP6
UIN
typedef EFI_STATUS(* IP6
OU
typedef struct
_IP6_FORM_CALLBACK_INFO IP6

Functions

EFI_STATUS EFIAPI Ip6
*Co
EFI_STATUS Ip6
IN
EFI_STATUS Ip6
VOID Ip6

EFI_STATUS Ip6ConfigD

Detailed Description

Definitions for EFI IPv6 Configuartion Protocol implementation.

Copyright (c) 2009 - 2013, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"paramtype">Attrib,
Bits) (BOOLEAN)((Attrib) & (Bits))

Referenced by Ip6ConfigCleanInstance(), Ip6ConfigReadConfigData(), and Ip6ConfigWriteConfigData().

#define DATA_ATTRIB_SIZE_FIXED 0x1

Referenced by Ip6ConfigCleanInstance(), Ip6ConfigInitInstance(), and Ip6ConfigReadConfigData().

#define DATA_ATTRIB_VOLATILE 0x2

Referenced by Ip6ConfigInitInstance(), and Ip6ConfigWriteConfigData().

#define IP6_CONFIG_DEFAULT_DAD_XMITS 1

Referenced by Ip6ConfigInitInstance().

#define IP6_CONFIG_DHCP6_OPTION_DNS_SERVERS 23

Referenced by Ip6ConfigParseDhcpReply(), Ip6ConfigStartStatefulAutoConfig(), and Ip6OnDADFinished().

#define IP6_CONFIG_DHCP6_OPTION_ORO 6

Referenced by Ip6ConfigStartStatefulAutoConfig(), and Ip6OnDADFinished().

#define IP6_CONFIG_INSTANCE_FROM_FORM_CALLBACK (Callback)

Value:

```
CR ((Callback), \
      IP6_CONFIG_INSTANCE, \
      CallbackInfo, \
      IP6_CONFIG_INSTANCE_SIGNATURE \
      )
```

Referenced by Ip6FormCallback(), and Ip6FormExtractConfig().

#define IP6_CONFIG_INSTANCE_FROM_PROTOCOL (Proto)

Value:

```
CR ((Proto), \
      IP6_CONFIG_INSTANCE, \
      Ip6Config, \
      IP6_CONFIG_INSTANCE_SIGNATURE \
      )
```

Referenced by EfiIp6ConfigGetData(), EfiIp6ConfigRegisterDataNotify(), EfiIp6ConfigSetData(), and EfiIp6ConfigUnregisterDataNotify().

#define IP6_CONFIG_INSTANCE_SIGNATURE SIGNATURE_32 ('I', 'P', '6', 'C')

Referenced by Ip6BuildNvAddressInfo(), Ip6ConfigInitInstance(), Ip6ConfigSetStatefulAddrCallback(), Ip6ConvertConfigNvDataToIfrNvData(), Ip6ConvertIfrNvDataToConfigNvDataAdvanced(), Ip6ConvertIfrNvDataToConfigNvDataGeneral(), and Ip6ManualAddrDadCallback().

#define IP6_CONFIG_VARIABLE_ATTRIBUTE (EFI_VARIABLE_NON_VOLATILE | EFI_VARIABLE_BOOTSERVICE_ACCESS)

Referenced by Ip6ConfigReadConfigData(), and Ip6ConfigWriteConfigData().

#define IP6_FORM_CALLBACK_INFO_FROM_CONFIG_ACCESS (ConfigAccess)

Value:

```
CR ((ConfigAccess), \
      IP6_FORM_CALLBACK_INFO, \
      HiiConfigAccess, \
      IP6_FORM_CALLBACK_INFO_SIGNATURE \
      )
```

Referenced by Ip6FormCallback(), and Ip6FormExtractConfig().

#define IP6_FORM_CALLBACK_INFO_SIGNATURE SIGNATURE_32 ('I', 'F', 'C', 'I')

Referenced by Ip6ConfigFormInit().

#define IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE (Instance)

Value:

```
CR ((Instance), \
      IP6_SERVICE, \
      Ip6ConfigInstance, \
      IP6_SERVICE_SIGNATURE \
```

)

Referenced by EfiIp6ConfigSetData(), Ip6ConfigDestroyDhcp6(), Ip6ConfigFormInit(), Ip6ConfigFormUnload(), Ip6ConfigGetIfInfo(), Ip6ConfigInitInstance(), Ip6ConfigOnDhcp6Event(), Ip6ConfigSetGateway(), Ip6ConfigSetMaunualAddress(), Ip6ConfigSetPolicy(), Ip6ConfigStartStatefulAutoConfig(), and Ip6ManualAddrDadCallback().

```
#define SET_DATA_ATTRIB ( Attrib,
                          Bits      )    ((Attrib) |= (Bits))
```

Referenced by Ip6ConfigInitInstance().

Typedef Documentation

typedef EFI_STATUS(* IP6_CONFIG_GET_DATA)(IN IP6_CONFIG_INSTANCE *Instance, IN OUT UINTN *DataSize, IN VOID *Data)

The prototype of work function for EfiIp6ConfigGetData().

Parameters:

[in]	*Instance*	The pointer to the IP6 config instance data.
[in,out]	*DataSize*	On input, in bytes, the size of Data. On output, in bytes, the size of buffer required to store the specified configuration data.
[in]	*Data*	The data buffer in which the configuration data is returned. Ignored if DataSize is ZERO.

Return values:

EFI_BUFFER_TOO_SMALL	The size of Data is too small for the specified configuration data, and the required size is returned in DataSize.
EFI_SUCCESS	The specified configuration data was obtained successfully.

typedef struct _IP6_CONFIG_INSTANCE IP6_CONFIG_INSTANCE

typedef EFI_STATUS(* IP6_CONFIG_SET_DATA)(IN IP6_CONFIG_INSTANCE *Instance, IN UINTN DataSize, IN VOID *Data)

The prototype of work function for EfiIp6ConfigSetData().

Parameters:

[in]	*Instance*	The pointer to the IP6 config instance data.
[in]	*DataSize*	In bytes, the size of the buffer pointed to by Data.
[in]	*Data*	The data buffer to set.

Return values:

EFI_BAD_BUFFER_SIZE	The DataSize does not match the size of the type, 8 bytes.
EFI_SUCCESS	The specified configuration data for the EFI IPv6 network stack was set successfully.

typedef struct _IP6_FORM_CALLBACK_INFO IP6_FORM_CALLBACK_INFO

Function Documentation

VOID Ip6ConfigCleanInstance (IN OUT IP6_CONFIG_INSTANCE * *Instance*)
Release an IP6_CONFIG_INSTANCE.

Parameters:

`[in,out]` *Instance* The buffer of IP6_CONFIG_INSTANCE to be freed.

References IP6_CONFIG_DATA_ITEM::Attribute, IP6_CONFIG_DATA_ITEM::Data, DATA_ATTRIB_SET, DATA_ATTRIB_SIZE_FIXED, IP6_CONFIG_DATA_ITEM::DataSize, Ip6ConfigDestroyDhcp6(), Ip6ConfigFormUnload(), and IP6_CONFIG_DATA::Ptr.

Referenced by Ip6CleanService().

EFI_STATUS Ip6ConfigDestroyDhcp6 (IN OUT IP6_CONFIG_INSTANCE * *Instance*)
Destroy the Dhcp6 child in IP6_CONFIG_INSTANCE and release the resources.

Parameters:

`[in,out]` *Instance* The buffer of IP6_CONFIG_INSTANCE to be freed.

Return values:

EFI_SUCCESS The child was successfully destroyed.
Others Failed to destroy the child.

References _IP6_SERVICE::Controller, _IP6_SERVICE::Image, and IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE.

Referenced by Ip6ConfigCleanInstance(), Ip6ConfigSetPolicy(), and Ip6DriverBindingStop().

EFI_STATUS Ip6ConfigInitInstance (OUT IP6_CONFIG_INSTANCE * *Instance*)
Initialize an IP6_CONFIG_INSTANCE.

Parameters:

`[out]` *Instance* The buffer of IP6_CONFIG_INSTANCE to be initialized.

Return values:

EFI_OUT_OF_RESOURCES Failed to allocate resources to complete the operation.
EFI_SUCCESS The IP6_CONFIG_INSTANCE initialized successfully.

References IP6_CONFIG_DATA_ITEM::Attribute, IP6_CONFIG_DATA_ITEM::Data, DATA_ATTRIB_SIZE_FIXED, DATA_ATTRIB_VOLATILE, IP6_CONFIG_DATA_ITEM::DataSize, EfiIp6ConfigGetData(), EfiIp6ConfigRegisterDataNotify(), EfiIp6ConfigSetData(), EfiIp6ConfigUnregisterDataNotify(), IP6_CONFIG_DATA_ITEM::GetData, _IP6_CONFIG_INSTANCE::IfIndex, IP6_CONFIG_DEFAULT_DAD_XMITS, IP6_CONFIG_INSTANCE_SIGNATURE, IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE, Ip6ConfigFormInit(), Ip6ConfigGetIfInfo(), Ip6ConfigInitIfInfo(), Ip6ConfigOnDhcp6Event(), Ip6ConfigReadConfigData(), Ip6ConfigSetAltIfId(), Ip6ConfigSetDadXmits(), Ip6ConfigSetDnsServer(), Ip6ConfigSetGateway(), Ip6ConfigSetMaunualAddress(), Ip6ConfigSetPolicy(), Ip6ConfigWriteConfigData(), _IP6_ADDRESS_INFO::Link, _IP6_SERVICE::MacString,

mIp6ConfigInstanceList, IP6_CONFIG_DATA::Ptr, SET_DATA_ATTRIB, IP6_CONFIG_DATA_ITEM::SetData, _IP6_SERVICE::SnpMode, and IP6_CONFIG_DATA_ITEM::Status.

Referenced by Ip6CreateService().

```
EFI_STATUS EFIAPI Ip6ConfigOnDhcp6Reply ( IN EFI_DHCP6_PROTOCOL *   This,
                                          IN VOID *                 Context,
                                          IN EFI_DHCP6_PACKET *     Packet
                                        )
```

The event process routine when the DHCPv6 server is answered with a reply packet for an information request.

Parameters:

- [in] *This* Points to the EFI_DHCP6_PROTOCOL.
- [in] *Context* The pointer to the IP6 configuration instance data.
- [in] *Packet* The DHCPv6 reply packet.

Return values:

- *EFI_SUCCESS* The DNS server address was retrieved from the reply packet.
- *EFI_NOT_READY* The reply packet does not contain the DNS server option, or the DNS server address is not valid.

The event process routine when the DHCPv6 server is answered with a reply packet for an information request.

Parameters:

- [in] *This* Points to the EFI_DHCP6_PROTOCOL.
- [in] *Context* The pointer to the IP6 configuration instance data.
- [in] *Packet* The DHCPv6 reply packet.

Return values:

- *EFI_SUCCESS* The DNS server address was retrieved from the reply packet.
- *EFI_NOT_READY* The reply packet does not contain the DNS server option, or the DNS server address is not valid.

References Ip6ConfigParseDhcpReply().

Referenced by Ip6ConfigStartStatefulAutoConfig(), and Ip6OnDADFinished().

```
EFI_STATUS Ip6ConfigStartStatefulAutoConfig ( IN IP6_CONFIG_INSTANCE *   Instance,
                                              IN BOOLEAN                 OtherInfoOnly
                                            )
```

The work function to trigger the DHCPv6 process to perform a stateful autoconfiguration.

Parameters:

- [in] *Instance* Pointer to the IP6 config instance data.
- [in] *OtherInfoOnly* If FALSE, get stateful address and other information via DHCPv6. Otherwise, only get the other information.

Return values:

EFI_SUCCESS	The operation finished successfully.
EFI_UNSUPPORTED	The DHCP6 driver is not available.

References _IP6_SERVICE::Controller, _IP6_SERVICE::Dhcp6NeedInfoRequest, _IP6_SERVICE::Dhcp6NeedStart, _IP6_SERVICE::Image, IP6_CONFIG_DHCP6_OPTION_DNS_SERVERS, IP6_CONFIG_DHCP6_OPTION_ORO, IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE, Ip6ConfigOnDhcp6Reply(), Ip6ConfigOnDhcp6SbInstalled(), and _IP6_SERVICE::LinkLocalOk.

Referenced by Ip6ConfigOnDhcp6SbInstalled(), and Ip6ProcessRouterAdvertise().

NetworkPkg/Ip6Dxe/Ip6ConfigNv.c File Reference

Functions

VOID EFIAPI Ip6ConfigManualAddressNotify (IN EFI_EVENT Event, IN VOID *Context)
EFI_STATUS Ip6ConfigNvGetData (IN EFI_IP6_CONFIG_PROTOCOL *Ip6Config, IN EFI_IP6_CONFIG_DATA_TYPE DataType, OUT UINTN *DataSize, OUT VOID **Data)
VOID Ip6FreeAddressInfoList (IN LIST_ENTRY *ListHead)
VOID Ip6ToStr (IN EFI_IPv6_ADDRESS *Ip6, OUT CHAR16 *Str)
EFI_STATUS Ip6ConvertInterfaceIdToString (OUT CHAR16 *String, IN EFI_IP6_CONFIG_INTERFACE_ID *IfId)
EFI_STATUS Ip6ParseInterfaceIdFromString (IN CONST CHAR16 *String, OUT EFI_IP6_CONFIG_INTERFACE_ID *IfId)
EFI_STATUS Ip6CreateOpCode (IN UINT16 StartLabelNumber, OUT VOID **StartOpCodeHandle, OUT EFI_IFR_GUID_LABEL **StartLabel, OUT VOID **EndOpCodeHandle, OUT EFI_IFR_GUID_LABEL **EndLabel)
EFI_STATUS Ip6ConvertAddressListToString (IN OUT CHAR16 *String, IN EFI_HII_HANDLE HiiHandle, IN IP6_CONFIG_NV_ADDRESS_TYPE AddressType, IN VOID *AddressInfo, IN UINTN AddressCount)
EFI_STATUS Ip6ParseAddressListFromString (IN CONST CHAR16 *String, OUT LIST_ENTRY *ListHead, OUT UINT32 *AddressCount)
EFI_STATUS Ip6ConvertInterfaceInfoToString (IN EFI_IP6_CONFIG_INTERFACE_INFO *IfInfo, IN EFI_HII_HANDLE HiiHandle, IN OUT IP6_CONFIG_IFR_NVDATA *IfrNvData)
EFI_STATUS Ip6BuildNvAddressInfo (IN IP6_CONFIG_INSTANCE *Instance, IN IP6_CONFIG_NV_ADDRESS_TYPE AddressType, OUT VOID **AddressInfo, OUT UINTN *AddressSize)
EFI_STATUS Ip6ConvertConfigNvDataToIfrNvData (IN OUT IP6_CONFIG_IFR_NVDATA *IfrNvData, IN IP6_CONFIG_INSTANCE *Instance)
EFI_STATUS Ip6ConvertIfrNvDataToConfigNvDataGeneral (IN IP6_CONFIG_IFR_NVDATA *IfrNvData, IN OUT IP6_CONFIG_INSTANCE *Instance)
EFI_STATUS Ip6ConvertIfrNvDataToConfigNvDataAdvanced (IN IP6_CONFIG_IFR_NVDATA *IfrNvData, IN OUT IP6_CONFIG_INSTANCE *Instance)
EFI_STATUS EFIAPI Ip6FormExtractConfig (IN CONST EFI_HII_CONFIG_ACCESS_PROTOCOL *This, IN CONST EFI_STRING Request, OUT EFI_STRING *Progress, OUT EFI_STRING *Results)
EFI_STATUS EFIAPI Ip6FormRouteConfig (IN CONST EFI_HII_CONFIG_ACCESS_PROTOCOL *This, IN CONST EFI_STRING Configuration, OUT EFI_STRING *Progress)
EFI_STATUS Ip6GetCurrentSetting (IN IP6_CONFIG_INSTANCE *Instance)
EFI_STATUS EFIAPI Ip6FormCallback (IN CONST EFI_HII_CONFIG_ACCESS_PROTOCOL *This, IN EFI_BROWSER_ACTION Action, IN EFI_QUESTION_ID QuestionId, IN UINT8 Type, IN EFI_IFR_TYPE_VALUE *Value, OUT EFI_BROWSER_ACTION_REQUEST *ActionRequest)
EFI_STATUS Ip6ConfigFormInit (IN OUT IP6_CONFIG_INSTANCE *Instance)
VOID Ip6ConfigFormUnload (IN OUT IP6_CONFIG_INSTANCE *Instance)

Variables

CHAR16 mIp6ConfigStorageName [] = L"IP6_CONFIG_IFR_NVDATA"

Detailed Description

Helper functions for configuring or obtaining the parameters relating to IP6.

Copyright (c) 2010 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS Ip6BuildNvAddressInfo ( IN IP6_CONFIG_INSTANCE *          Instance,
                                   IN IP6_CONFIG_NV_ADDRESS_TYPE     AddressType,
                                   OUT VOID **                       AddressInfo,
                                   OUT UINTN *                       AddressSize
                                 )
```

Build the address info list from list array of node in IP6_ADDRESS_INFO_ENTRY.

Parameters:

[in]	*Instance*	Points to IP6 config instance data.
[in]	*AddressType*	The address type.
[out]	*AddressInfo*	The pointer to the buffer to store the address list.
[out]	*AddressSize*	The address size of the address list.

Return values:

EFI_SUCCESS	The operation finished successfully.
EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_UNSUPPORTED	The AddressType is not supported.

References IP6_ADDRESS_INFO_ENTRY::AddrInfo, IP6_CONFIG_NVDATA::DnsAddress, IP6_CONFIG_NVDATA::DnsAddressCount, IP6_CONFIG_NVDATA::GatewayAddress, IP6_CONFIG_NVDATA::GatewayAddressCount, IP6_CONFIG_INSTANCE_SIGNATURE, Ip6ConfigNvDnsAddress, Ip6ConfigNvGatewayAddress, Ip6ConfigNvHostAddress, IP6_CONFIG_NVDATA::ManualAddress, and IP6_CONFIG_NVDATA::ManualAddressCount.

Referenced by Ip6ConvertIfrNvDataToConfigNvDataAdvanced().

```
EFI_STATUS Ip6ConfigFormInit ( IN OUT IP6_CONFIG_INSTANCE *   Instance   )
```

Install HII Config Access protocol for network device and allocate resources.

Parameters:

[in,out]	*Instance*	The IP6_CONFIG_INSTANCE to create a form.

Return values:

EFI_SUCCESS	The HII Config Access protocol is installed.

EFI_OUT_OF_RESOURCES	Failed to allocate memory.
Others	Other errors as indicated.

References _IP6_FORM_CALLBACK_INFO::ChildHandle, _IP6_SERVICE::Controller, gIp6ConfigNvDataGuid, _IP6_FORM_CALLBACK_INFO::HiiConfigAccess, _IP6_FORM_CALLBACK_INFO::HiiVendorDevicePath, _IP6_SERVICE::Image, IP6_FORM_CALLBACK_INFO_SIGNATURE, IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE, Ip6ConfigBin, Ip6ConfigFormUnload(), Ip6DxeStrings, Ip6FormCallback(), Ip6FormExtractConfig(), Ip6FormRouteConfig(), _IP6_FORM_CALLBACK_INFO::RegisteredHandle, and _IP6_FORM_CALLBACK_INFO::Signature.

Referenced by Ip6ConfigInitInstance().

VOID Ip6ConfigFormUnload (IN OUT IP6_CONFIG_INSTANCE * *Instance*)

Uninstall the HII Config Access protocol for network devices and free up the resources.

Parameters:

`[in,out]` *Instance*	The IP6_CONFIG_INSTANCE to unload a form.

References _IP6_FORM_CALLBACK_INFO::ChildHandle, _IP6_SERVICE::Controller, IP6_CONFIG_NVDATA::DnsAddress, IP6_CONFIG_NVDATA::DnsAddressCount, IP6_CONFIG_NVDATA::GatewayAddress, IP6_CONFIG_NVDATA::GatewayAddressCount, _IP6_FORM_CALLBACK_INFO::HiiConfigAccess, _IP6_FORM_CALLBACK_INFO::HiiVendorDevicePath, _IP6_SERVICE::Image, IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE, Ip6FreeAddressInfoList(), IP6_CONFIG_NVDATA::ManualAddress, IP6_CONFIG_NVDATA::ManualAddressCount, and _IP6_FORM_CALLBACK_INFO::RegisteredHandle.

Referenced by Ip6ConfigCleanInstance(), and Ip6ConfigFormInit().

VOID EFIAPI Ip6ConfigManualAddressNotify (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

The notify function of create event when performing a manual configuration.

Parameters:

`[in]` *Event*	The pointer of Event.
`[in]` *Context*	The pointer of Context.

Referenced by Ip6ConvertIfrNvDataToConfigNvDataAdvanced().

EFI_STATUS Ip6ConfigNvGetData (IN EFI_IP6_CONFIG_PROTOCOL * *Ip6Config*,
IN EFI_IP6_CONFIG_DATA_TYPE *DataType*,
OUT UINTN * *DataSize*,
OUT VOID ** *Data*
)

Get the configuration data for the EFI IPv6 network stack running on the communication. It is a help function to the call EfiIp6ConfigGetData().

Parameters:

[in]	*Ip6Config*	The pointer to the EFI_IP6_CONFIG_PROTOCOL instance.
[in]	*DataType*	The type of data to get.
[out]	*DataSize*	The size of buffer required in bytes.
[out]	*Data*	The data buffer in which the configuration data is returned. The type of the data buffer associated with the DataType. It is the caller's responsibility to free the resource.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_INVALID_PARAMETER	One or more of the followings are TRUE: ◊ Ip6Config is NULL or invalid. ◊ DataSize is NULL. ◊ Data is NULL.
EFI_OUT_OF_RESOURCES	Fail to perform the operation due to lack of resources.
EFI_NOT_READY	The specified configuration data is not ready due to an asynchronous configuration process already in progress.
EFI_NOT_FOUND	The specified configuration data was not found.

Referenced by Ip6ConvertConfigNvDataToIfrNvData(), and Ip6GetCurrentSetting().

```
EFI_STATUS
Ip6ConvertAddressListToString   ( IN OUT CHAR16 *                  String,
                                  IN EFI_HII_HANDLE                HiiHandle,
                                  IN IP6_CONFIG_NV_ADDRESS_TYPE    AddressType,
                                  IN VOID *                        AddressInfo,
                                  IN UINTN                         AddressCount
                                  )
```

This function converts the different format of address list to string format and then generates the corresponding text opcode to illustarate the address info in IP6 configuration page. Currently, the following formats are supported: EFI_IP6_ADDRESS_INFO AddressType: Ip6ConfigNvHostAddress; EFI_IPv6_ADDRESS AddressType: Ip6ConfigNvGatewayAddress and Ip6ConfigNvDnsAddress; EFI_IP6_ROUTE_TABLE AddressType: Ip6ConfigNvRouteTable.

Parameters:

[in,out]	*String*	The pointer to the buffer to store the converted string.
[in]	*HiiHandle*	A handle that was previously registered in the HII Database.
[in]	*AddressType*	The address type.
[in]	*AddressInfo*	Pointer to the address list.
[in]	*AddressCount*	The address count of the address list.

Return values:

EFI_SUCCESS	The operation finished successfully.
EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_UNSUPPORTED	The AddressType is not supported.

References DNS_ADDRESS_LABEL, FORMID_MAIN_FORM, GATEWAY_ADDRESS_LABEL, gIp6ConfigNvDataGuid, HOST_ADDRESS_LABEL, IP6_ADDRESS_DELIMITER, Ip6ConfigNvDnsAddress, Ip6ConfigNvGatewayAddress, Ip6ConfigNvHostAddress, Ip6ConfigNvRouteTable, Ip6CreateOpCode(), Ip6ToStr(), and ROUTE_TABLE_LABEL.

Referenced by Ip6GetCurrentSetting().

EFI_STATUS Ip6ConvertConfigNvDataToIfrNvData (	IN OUT IP6_CONFIG_IFR_NVDATA *	*IfrNvData*,
	IN IP6_CONFIG_INSTANCE *	*Instance*
)		

Convert the IP6 configuration data into the IFR data.

Parameters:

`[in,out]`	*IfrNvData*	The IFR NV data.
`[in]`	*Instance*	The IP6 config instance data.

Return values:

EFI_SUCCESS	The operation finished successfully.
EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_UNSUPPORTED	The policy is not supported in the current implementation.
Others	Other errors as indicated.

References IP6_CONFIG_NVDATA::InterfaceId, IP6_CONFIG_INSTANCE_SIGNATURE, IP6_POLICY_AUTO, IP6_POLICY_MANUAL, Ip6ConfigNvGetData(), Ip6ConvertInterfaceIdToString(), Ip6ConvertInterfaceInfoToString(), and Policy.

Referenced by Ip6FormExtractConfig().

EFI_STATUS Ip6ConvertIfrNvDataToConfigNvDataAdvanced (	IN IP6_CONFIG_IFR_NVDATA *	*IfrNvData*,
	IN OUT IP6_CONFIG_INSTANCE *	*Instance*
)		

Convert IFR data into IP6 configuration data. The policy, configured manual address, gateway address, and DNS server address will be saved.

Parameters:

`[in]`	*IfrNvData*	The IFR NV data.
`[in,out]`	*Instance*	The IP6 config instance data.

Return values:

EFI_SUCCESS	The operation finished successfully.
EFI_INVALID_PARAMETER	Any input parameter is invalid.
Others	Other errors as indicated.

References IP6_CONFIG_NVDATA::DnsAddress, IP6_CONFIG_NVDATA::DnsAddressCount, IP6_CONFIG_NVDATA::GatewayAddress, IP6_CONFIG_NVDATA::GatewayAddressCount, IP6_CONFIG_INSTANCE_SIGNATURE, IP6_POLICY_AUTO, Ip6BuildNvAddressInfo(), Ip6ConfigManualAddressNotify(), Ip6ConfigNvDnsAddress, Ip6ConfigNvGatewayAddress, Ip6ConfigNvHostAddress, IP6_CONFIG_NVDATA::ManualAddress, IP6_CONFIG_NVDATA::ManualAddressCount, and IP6_CONFIG_NVDATA::Policy.

Referenced by Ip6FormCallback().

```
EFI_STATUS
Ip6ConvertIfrNvDataToConfigNvDataGeneral ( IN IP6_CONFIG_IFR_NVDATA *  IfrNvData,
                                           IN OUT IP6_CONFIG_INSTANCE *  Instance
                                         )
```

Convert IFR data into IP6 configuration data. The policy, alternative interface ID, and DAD transmit counts, and will be saved.

Parameters:

[in]	*IfrNvData*	The IFR NV data.
[in,out]	*Instance*	The IP6 config instance data.

Return values:

EFI_SUCCESS	The operation finished successfully.
EFI_INVALID_PARAMETER	Any input parameter is invalid.
Others	Other errors as indicated.

References IP6_CONFIG_NVDATA::DadTransmitCount, IP6_CONFIG_NVDATA::InterfaceId, IP6_CONFIG_INSTANCE_SIGNATURE, IP6_POLICY_AUTO, IP6_POLICY_MANUAL, and IP6_CONFIG_NVDATA::Policy.

Referenced by Ip6FormCallback().

```
EFI_STATUS Ip6ConvertInterfaceIdToString ( OUT CHAR16 *                    String,
                                           IN EFI_IP6_CONFIG_INTERFACE_ID *  IfId
                                         )
```

Convert EFI_IP6_CONFIG_INTERFACE_ID to string format.

Parameters:

[out]	*String*	The buffer to store the converted string.
[in]	*IfId*	The pointer of EFI_IP6_CONFIG_INTERFACE_ID.

Return values:

EFI_SUCCESS	The string converted successfully.
EFI_INVALID_PARAMETER	Any input parameter is invalid.

References INTERFACE_ID_STR_STORAGE.

Referenced by Ip6ConvertConfigNvDataToIfrNvData().

```
EFI_STATUS
Ip6ConvertInterfaceInfoToString ( IN EFI_IP6_CONFIG_INTERFACE_INFO *  IfInfo,
                                  IN EFI_HII_HANDLE                   HiiHandle,
                                  IN OUT IP6_CONFIG_IFR_NVDATA *      IfrNvData
                                )
```

This function converts the interface info to string and draws it to the IP6 UI. The interface information includes interface name, interface type, hardware address and route table information.

Parameters:

[in]	*IfInfo*	The pointer of EFI_IP6_CONFIG_INTERFACE_INFO.

[in]	*HiiHandle*	The handle that was previously registered in the HII Database.
[in,out]	*IfrNvData*	Points to IP6_CONFIG_IFR_NVDATA.

Return values:

EFI_SUCCESS	The operation finished successfully.
EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_OUT_OF_RESOURCES	The operation failed due to lack of resources.

References ADDRESS_STR_MAX_SIZE, IP6_ETHERNET, IP6_EXPERIMENTAL_ETHERNET, Ip6InterfaceTypeEthernet, and Ip6InterfaceTypeExperimentalEthernet.

Referenced by Ip6ConvertConfigNvDataToIfrNvData().

```
EFI_STATUS Ip6CreateOpCode ( IN UINT16                    StartLabelNumber,
                             OUT VOID **                  StartOpCodeHandle,
                             OUT EFI_IFR_GUID_LABEL **    StartLabel,
                             OUT VOID **                  EndOpCodeHandle,
                             OUT EFI_IFR_GUID_LABEL **    EndLabel
                           )
```

Create Hii Extend Label OpCode as the start opcode and end opcode. It is a help function.

Parameters:

[in]	*StartLabelNumber*	The number of start label.
[out]	*StartOpCodeHandle*	Points to the start opcode handle.
[out]	*StartLabel*	Points to the created start opcode.
[out]	*EndOpCodeHandle*	Points to the end opcode handle.
[out]	*EndLabel*	Points to the created end opcode.

Return values:

EFI_OUT_OF_RESOURCES	Does not have sufficient resources to finish this operation.
EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_SUCCESS	The operation completed successfully.

References LABEL_END.

Referenced by Ip6ConvertAddressListToString().

```
EFI_STATUS EFIAPI
Ip6FormCallback ( IN CONST
                  EFI_HII_CONFIG_ACCESS_PROTOCOL *   This,
                  IN EFI_BROWSER_ACTION              Action,
                  IN EFI_QUESTION_ID                 QuestionId,
                  IN UINT8                           Type,
                  IN EFI_IFR_TYPE_VALUE *            Value,
                  OUT EFI_BROWSER_ACTION_REQUEST *   ActionRequest
                )
```

This function is called to provide results data to the driver. This data consists of a unique key that is used to identify which data is either being passed back or being asked for.

Parameters:

[in]	*This*	Points to the EFI_HII_CONFIG_ACCESS_PROTOCOL.
[in]	*Action*	Specifies the type of action taken by the browser.
[in]	*QuestionId*	A unique value which is sent to the original exporting driver so that it can identify the type of data to expect. The format of the data tends to vary based on the opcode that generated the callback.
[in]	*Type*	The type of value for the question.
[in]	*Value*	A pointer to the data being sent to the original exporting driver.
[out]	*ActionRequest*	On return, points to the action requested by the callback function.

Return values:

EFI_SUCCESS	The callback successfully handled the action.
EFI_OUT_OF_RESOURCES	Not enough storage is available to hold the variable and its data.
EFI_DEVICE_ERROR	The variable could not be saved.
EFI_UNSUPPORTED	The specified Action is not supported by the callback. Currently not implemented.
EFI_INVALID_PARAMETER	Passed in the wrong parameter.
Others	Other errors as indicated.

References IP6_CONFIG_NVDATA::DnsAddress, IP6_CONFIG_NVDATA::DnsAddressCount, IP6_CONFIG_NVDATA::GatewayAddress, IP6_CONFIG_NVDATA::GatewayAddressCount, IP6_CONFIG_NVDATA::InterfaceId, IP6_CONFIG_INSTANCE_FROM_FORM_CALLBACK, IP6_FORM_CALLBACK_INFO_FROM_CONFIG_ACCESS, Ip6ConvertIfrNvDataToConfigNvDataAdvanced(), Ip6ConvertIfrNvDataToConfigNvDataGeneral(), Ip6FreeAddressInfoList(), Ip6GetCurrentSetting(), _IP6_CONFIG_INSTANCE::Ip6NvData, Ip6ParseAddressListFromString(), Ip6ParseInterfaceIdFromString(), KEY_DNS_ADDRESS, KEY_GATEWAY_ADDRESS, KEY_GET_CURRENT_SETTING, KEY_IGNORE_CONFIG_CHANGES, KEY_INTERFACE_ID, KEY_MANUAL_ADDRESS, KEY_SAVE_CHANGES, KEY_SAVE_CONFIG_CHANGES, IP6_CONFIG_NVDATA::ManualAddress, and IP6_CONFIG_NVDATA::ManualAddressCount.

Referenced by Ip6ConfigFormInit().

```
EFI_STATUS EFIAPI
Ip6FormExtractConfig (
  IN CONST EFI_HII_CONFIG_ACCESS_PROTOCOL *   This,
  IN CONST EFI_STRING                         Request,
  OUT EFI_STRING *                            Progress,
  OUT EFI_STRING *                            Results
)
```

This function allows the caller to request the current configuration for one or more named elements. The resulting string is in <ConfigAltResp> format. Any and all alternative configuration strings shall also be appended to the end of the current configuration string. If they are, they must appear after the current configuration. They must contain the same routing (GUID, NAME, PATH) as the current configuration string. They must have an additional description indicating the type of alternative configuration the string represents, "ALTCFG=<StringToken>". That <StringToken> (when converted from Hex UNICODE to binary) is a reference to a string in the associated string pack.

Parameters:

[in]	*This*	Points to the EFI_HII_CONFIG_ACCESS_PROTOCOL.
[in]	*Request*	A null-terminated Unicode string in <ConfigRequest> format. Note that this includes the routing information as well as the configurable name / value pairs. It is invalid for this string to be in <MultiConfigRequest> format.

[out]	*Progress*	On return, points to a character in the Request string. Points to the string's null terminator if request was successful. Points to the most recent "&" before the first failing name / value pair (or the beginning of the string if the failure is in the first name / value pair) if the request was not successful.
[out]	*Results*	A null-terminated Unicode string in <ConfigAltResp> format which has all values filled in for the names in the Request string. String to be allocated by the called function.

Return values:

EFI_SUCCESS	The Results string is filled with the values corresponding to all requested names.
EFI_OUT_OF_RESOURCES	Not enough memory to store the parts of the results that must be stored awaiting possible future protocols.
EFI_INVALID_PARAMETER	For example, passing in a NULL for the Request parameter would result in this type of error. In this case, the Progress parameter would be set to NULL.
EFI_NOT_FOUND	Routing data doesn't match any known driver. Progress set to the first character in the routing header. Note: There is no requirement that the driver validate the routing data. It must skip the <ConfigHdr> in order to process the names.
EFI_INVALID_PARAMETER	Illegal syntax. Progress set to most recent & before the error or the beginning of the string.
EFI_INVALID_PARAMETER	Unknown name. Progress points to the & before the name in question. Currently not implemented.

References _IP6_FORM_CALLBACK_INFO::ChildHandle, gIp6ConfigNvDataGuid, IP6_CONFIG_INSTANCE_FROM_FORM_CALLBACK, IP6_FORM_CALLBACK_INFO_FROM_CONFIG_ACCESS, Ip6ConvertConfigNvDataToIfrNvData(), and mIp6ConfigStorageName.

Referenced by Ip6ConfigFormInit().

```
EFI_STATUS EFIAPI
Ip6FormRouteConfig (
  IN CONST EFI_HII_CONFIG_ACCESS_PROTOCOL *  This,
  IN CONST EFI_STRING                        Configuration,
  OUT EFI_STRING *                           Progress
)
```

This function applies changes in a driver's configuration. Input is a Configuration, which has the routing data for this driver followed by name / value configuration pairs. The driver must apply those pairs to its configurable storage. If the driver's configuration is stored in a linear block of data and the driver's name / value pairs are in <BlockConfig> format, it may use the ConfigToBlock helper function (above) to simplify the job. Currently not implemented.

Parameters:

[in]	*This*	Points to the EFI_HII_CONFIG_ACCESS_PROTOCOL.
[in]	*Configuration*	A null-terminated Unicode string in <ConfigString> format.
[out]	*Progress*	A pointer to a string filled in with the offset of the most recent '&' before the first failing name / value pair (or the beginn ing of the string if the failure is in the first name / value pair) or the terminating NULL if all was successful.

Return values:

EFI_SUCCESS	The results have been distributed or are awaiting distribution.

EFI_OUT_OF_MEMORY	Not enough memory to store the parts of the results that must be stored awaiting possible future protocols.
EFI_INVALID_PARAMETERS	Passing in a NULL for the Results parameter would result in this type of error.
EFI_NOT_FOUND	Target for the specified routing data was not found.

References gIp6ConfigNvDataGuid, and mIp6ConfigStorageName.

Referenced by Ip6ConfigFormInit().

VOID Ip6FreeAddressInfoList (IN LIST_ENTRY * *ListHead*)

Free all nodes in IP6_ADDRESS_INFO_ENTRY in the list array specified with ListHead.

Parameters:

`[in]` *ListHead* The head of the list array in IP6_ADDRESS_INFO_ENTRY.

References IP6_ADDRESS_INFO_ENTRY::Link.

Referenced by Ip6ConfigFormUnload(), Ip6FormCallback(), and Ip6ParseAddressListFromString().

EFI_STATUS Ip6GetCurrentSetting (IN IP6_CONFIG_INSTANCE * *Instance*)

Display host addresses, route table, DNS addresses and gateway addresses in "IPv6 Current Setting" page.

Parameters:

`[in]` *Instance* The IP6 config instance data.

Return values:

EFI_SUCCESS	The operation finished successfully.
Others	Other errors as indicated.

References ADDRESS_STR_MAX_SIZE, Ip6ConfigNvDnsAddress, Ip6ConfigNvGatewayAddress, Ip6ConfigNvGetData(), Ip6ConfigNvHostAddress, Ip6ConfigNvRouteTable, and Ip6ConvertAddressListToString().

Referenced by Ip6FormCallback().

EFI_STATUS Ip6ParseAddressListFromString (IN CONST CHAR16 * *String*, OUT LIST_ENTRY * *ListHead*, OUT UINT32 * *AddressCount*)

Parse address list in string format and convert it to a list array of node in IP6_ADDRESS_INFO_ENTRY.

Parameters:

`[in]`	*String*	The buffer to string to be parsed.
`[out]`	*ListHead*	The list head of array.
`[out]`	*AddressCount*	The number of list nodes in the array.

Return values:

EFI_SUCCESS	The operation finished successfully.
EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_OUT_OF_RESOURCES	Failed to perform the operation due to lack of resource.

References IP6_ADDRESS_INFO_ENTRY::AddrInfo, IP6_ADDRESS_DELIMITER, Ip6FreeAddressInfoList(), and IP6_ADDRESS_INFO_ENTRY::Link.

Referenced by Ip6FormCallback().

```
EFI_STATUS Ip6ParseInterfaceIdFromString ( IN CONST CHAR16 *                    String,
                                           OUT EFI_IP6_CONFIG_INTERFACE_ID *    IfId
                                         )
```

Parse InterfaceId in string format and convert it to EFI_IP6_CONFIG_INTERFACE_ID.

Parameters:

[in]	*String*	The buffer of the string to be parsed.
[out]	*IfId*	The pointer of EFI_IP6_CONFIG_INTERFACE_ID.

Return values:

EFI_SUCCESS	The operation finished successfully.
EFI_INVALID_PARAMETER	Any input parameter is invalid.

Referenced by Ip6FormCallback().

```
VOID Ip6ToStr ( IN EFI_IPv6_ADDRESS *   Ip6,
                OUT CHAR16 *            Str
              )
```

Convert the IPv6 address into a formatted string.

Parameters:

[in]	*Ip6*	The IPv6 address.
[out]	*Str*	The formatted IP string.

References IP6_STR_MAX_SIZE.

Referenced by Ip6ConvertAddressListToString().

Variable Documentation

```
CHAR16 mIp6ConfigStorageName[] = L"IP6_CONFIG_IFR_NVDATA"
```

Referenced by Ip6FormExtractConfig(), and Ip6FormRouteConfig().

- File List
- Globals

NetworkPkg/Ip6Dxe/Ip6ConfigNv.h File Reference

Defines

"memItemRight" valign="bottom">IP6_ETHERNET L"Ethernet"

#define IP6_EXPERIMENTAL_ETHERNET L"Experim
Ethernet"

#define IP6_ADDRESS_DELIMITER L' '

#define IP6_LINK_LOCAL_PREFIX L"FE80::"

Enumerations

enum IP6_INTERFACE_TYPE {
Ip6InterfaceTypeEthernet = 1,
Ip6InterfaceTypeExperimentalEthernet }

enum IP6_CONFIG_NV_ADDRESS_TYPE {
Ip6ConfigNvHostAddress,
Ip6ConfigNvGatewayAddress,
Ip6ConfigNvDnsAddress,
Ip6ConfigNvRouteTable }

Functions

EFI_STATUS Ip6ConfigFormInit (IN OUT
IP6_CONFIG_INSTANCE *Instance)

VOID Ip6ConfigFormUnload (IN OUT
IP6_CONFIG_INSTANCE *Instance)

Variables

UINT8 Ip6ConfigBin []

UINT8 Ip6DxeStrings []

Detailed Description

The header file of Ip6ConfigNv.c.

Copyright (c) 2010 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Ip6ConvertAddressListToString(), and Ip6ParseAddressListFromString().

#define IP6_ETHERNET L"Ethernet"

Referenced by Ip6ConvertInterfaceInfoToString().

#define IP6_EXPERIMENTAL_ETHERNET L"Experimental Ethernet"

Referenced by Ip6ConvertInterfaceInfoToString().

#define IP6_LINK_LOCAL_PREFIX L"FE80::"

Enumeration Type Documentation

enum IP6_CONFIG_NV_ADDRESS_TYPE

Enumerator:

Ip6ConfigNvHostAddress
Ip6ConfigNvGatewayAddress
Ip6ConfigNvDnsAddress
Ip6ConfigNvRouteTable

enum IP6_INTERFACE_TYPE

Enumerator:

Ip6InterfaceTypeEthernet
Ip6InterfaceTypeExperimentalEthernet

Function Documentation

EFI_STATUS Ip6ConfigFormInit (IN OUT IP6_CONFIG_INSTANCE * *Instance*)

Install HII Config Access protocol for network device and allocate resources.

Parameters:

`[in,out]` *Instance* The IP6_CONFIG_INSTANCE to create a form.

Return values:

EFI_SUCCESS	The HII Config Access protocol is installed.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
Others	Other errors as indicated.

References _IP6_FORM_CALLBACK_INFO::ChildHandle, _IP6_SERVICE::Controller, gIp6ConfigNvDataGuid, _IP6_FORM_CALLBACK_INFO::HiiConfigAccess, _IP6_FORM_CALLBACK_INFO::HiiVendorDevicePath, _IP6_SERVICE::Image, IP6_FORM_CALLBACK_INFO_SIGNATURE, IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE, Ip6ConfigBin, Ip6ConfigFormUnload(), Ip6DxeStrings, Ip6FormCallback(), Ip6FormExtractConfig(), Ip6FormRouteConfig(), _IP6_FORM_CALLBACK_INFO::RegisteredHandle, and _IP6_FORM_CALLBACK_INFO::Signature.

Referenced by Ip6ConfigInitInstance().

VOID Ip6ConfigFormUnload (IN OUT IP6_CONFIG_INSTANCE * *Instance*)

Uninstall HII Config Access protocol for network device and free resource.

Parameters:

`[in,out]` *Instance* The IP6_CONFIG_INSTANCE to unload a form.

Uninstall the HII Config Access protocol for network devices and free up the resources.

Parameters:

`[in,out]` *Instance* The IP6_CONFIG_INSTANCE to unload a form.

References _IP6_FORM_CALLBACK_INFO::ChildHandle, _IP6_SERVICE::Controller, IP6_CONFIG_NVDATA::DnsAddress, IP6_CONFIG_NVDATA::DnsAddressCount, IP6_CONFIG_NVDATA::GatewayAddress, IP6_CONFIG_NVDATA::GatewayAddressCount, _IP6_FORM_CALLBACK_INFO::HiiConfigAccess, _IP6_FORM_CALLBACK_INFO::HiiVendorDevicePath, _IP6_SERVICE::Image, IP6_SERVICE_FROM_IP6_CONFIG_INSTANCE, Ip6FreeAddressInfoList(), IP6_CONFIG_NVDATA::ManualAddress, IP6_CONFIG_NVDATA::ManualAddressCount, and _IP6_FORM_CALLBACK_INFO::RegisteredHandle.

Referenced by Ip6ConfigCleanInstance(), and Ip6ConfigFormInit().

Variable Documentation

UINT8 Ip6ConfigBin[]

Referenced by Ip6ConfigFormInit().

UINT8 Ip6DxeStrings[]

Referenced by Ip6ConfigFormInit().

NetworkPkg/Ip6Dxe/Ip6Driver.c File Reference

Functions

VOID EFIAPI IpSec2InstalledCallback (IN EFI_EVENT Event, IN VOID *Context)
EFI_STATUS EFIAPI Ip6DriverEntryPoint (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)
EFI_STATUS EFIAPI Ip6DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS Ip6CleanService (IN IP6_SERVICE *IpSb)
EFI_STATUS Ip6CreateService (IN EFI_HANDLE Controller, IN EFI_HANDLE ImageHandle, OUT IP6_SERVICE **Service)
EFI_STATUS EFIAPI Ip6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI Ip6DestroyChildEntryInHandleBuffer (IN LIST_ENTRY *Entry, IN VOID *Context)
EFI_STATUS EFIAPI Ip6DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)
EFI_STATUS EFIAPI Ip6ServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE *ChildHandle)
EFI_STATUS EFIAPI Ip6ServiceBindingDestroyChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE ChildHandle)

Variables

EFI_DRIVER_BINDING_PROTOCOL gIp6DriverBinding
BOOLEAN mIpSec2Installed = FALSE

Detailed Description

The driver binding and service binding protocol for IP6 driver.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.
(C) Copyright 2015 Hewlett-Packard Development Company, L.P.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS Ip6CleanService (IN IP6_SERVICE * *IpSb*)

Clean up an IP6 service binding instance. It releases all the resource allocated by the instance. The instance may be partly initialized, or partly destroyed. If a resource is destroyed, it is marked as that in case the destroy failed and being called again later.

Parameters:

`[in]` *IpSb* The IP6 service binding instance to clean up.

Return values:

EFI_SUCCESS The resource used by the instance are cleaned up.
Others Failed to clean up some of the resources.

References IP6_LINK_LOCAL_SCOPE, Ip6CleanAssembleTable(), Ip6CleanDefaultRouterList(), Ip6CleanInterface(), Ip6CleanPrefixListTable(), Ip6CleanRouteTable(), Ip6ConfigCleanInstance(), Ip6FreeNeighborEntry(), Ip6LeaveGroup(), and Ip6SetToAllNodeMulticast().

Referenced by Ip6CreateService(), Ip6DriverBindingStart(), and Ip6DriverBindingStop().

EFI_STATUS Ip6CreateService (IN EFI_HANDLE *Controller*,
IN EFI_HANDLE *ImageHandle*,
OUT IP6_SERVICE ** *Service*
)

Create a new IP6 driver service binding protocol.

Parameters:

`[in]` *Controller* The controller that has MNP service binding installed.
`[in]` *ImageHandle* The IP6 driver's image handle.
`[out]` *Service* The variable to receive the newly created IP6 service.

Return values:

EFI_OUT_OF_RESOURCES Failed to allocate some resources.
EFI_SUCCESS A new IP6 service binding private is created.

References _IP6_SERVICE::Assemble, _IP6_SERVICE::AutonomousPrefix, _IP6_SERVICE::BaseReachableTime, IP6_LINK_RX_TOKEN::CallBack, _IP6_SERVICE::Children, IP6_LINK_RX_TOKEN::Context, _IP6_SERVICE::Controller, _IP6_SERVICE::CurHopLimit, IP6_CONFIG_DATA_ITEM::Data, _IP6_CONFIG_INSTANCE::DataItem, IP6_CONFIG_DATA_ITEM::DataSize, _IP6_SERVICE::DefaultInterface, _IP6_SERVICE::DefaultRouterList, _IP6_SERVICE::Dhcp6NeedInfoRequest, _IP6_SERVICE::Dhcp6NeedStart, _IP6_SERVICE::FasterTimer, IP6_MLD_SERVICE_DATA::Groups, _IP6_SERVICE::Image, _IP6_SERVICE::InterfaceId, _IP6_SERVICE::InterfaceIdLen, _IP6_SERVICE::Interfaces, IP6_ETHER_PROTO, IP6_HOP_LIMIT, IP6_IF_ID_LEN, IP6_LINK_RX_SIGNATURE, IP6_MAX_RTR_SOLICITATIONS, IP6_MIN_LINK_MTU, IP6_REACHABLE_TIME, IP6_RETRANS_TIMER, IP6_SERVICE_SIGNATURE, IP6_SERVICE_UNSTARTED, Ip6CleanService(), Ip6ConfigInitInstance(), _IP6_SERVICE::Ip6ConfigInstance, Ip6CreateAssembleTable(), Ip6CreateInterface(), Ip6CreateRouteTable(), Ip6InitMld(), Ip6NdFasterTimerTicking(), Ip6OnFrameReceived(), Ip6ServiceBindingCreateChild(), Ip6ServiceBindingDestroyChild(), Ip6ServiceConfigMnp(),

Ip6TimerTicking(), Ip6UpdateReachableTime(), _IP6_INTERFACE::Link, _IP6_SERVICE::LinkLocalAddr, _IP6_SERVICE::LinkLocalDadFail, _IP6_SERVICE::LinkLocalOk, _IP6_SERVICE::LinkMTU, _IP6_SERVICE::MacString, _IP6_SERVICE::MaxPacketSize, _IP6_SERVICE::MldCtrl, IP6_MLD_SERVICE_DATA::Mldv1QuerySeen, _IP6_SERVICE::Mnp, _IP6_SERVICE::MnpChildHandle, _IP6_SERVICE::MnpConfigData, IP6_LINK_RX_TOKEN::MnpToken, _IP6_SERVICE::NeighborTable, _IP6_SERVICE::NumChildren, _IP6_SERVICE::OldMaxPacketSize, _IP6_SERVICE::OnlinkPrefix, IP6_CONFIG_DATA::Ptr, _IP6_SERVICE::RecvRequest, _IP6_SERVICE::RetransTimer, _IP6_SERVICE::RoundRobin, _IP6_SERVICE::RouterAdvertiseReceived, _IP6_SERVICE::RouteTable, _IP6_SERVICE::ServiceBinding, IP6_CONFIG_DATA_ITEM::SetData, IP6_LINK_RX_TOKEN::Signature, _IP6_SERVICE::Signature, _IP6_SERVICE::SnpMode, _IP6_SERVICE::SolicitTimer, _IP6_SERVICE::State, _IP6_SERVICE::Ticks, and _IP6_SERVICE::Timer.

Referenced by Ip6DriverBindingStart().

```
EFI_STATUS EFIAPI Ip6DestroyChildEntryInHandleBuffer (  IN LIST_ENTRY *  Entry,
                                                        IN VOID *        Context
                                                      )
```

Callback function which provided by user to remove one node in NetDestroyLinkList process.

Parameters:

[in]	*Entry*	The entry to be removed.
[in]	*Context*	Pointer to the callback context corresponds to the Context in NetDestroyLinkList.

Return values:

EFI_SUCCESS	The entry has been removed successfully.
Others	Fail to remove the entry.

References _IP6_PROTOCOL::Handle, and IP6_PROTOCOL_SIGNATURE.

Referenced by Ip6DriverBindingStop().

```
EFI_STATUS EFIAPI Ip6DriverBindingStart (  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
                                           IN EFI_HANDLE                     ControllerHandle,
                                           IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
                                         )
```

Start this driver on ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to bind driver to.
[in]	*RemainingDevicePath*	Optional parameter used to pick a specific child device to start.

Return values:

EFI_SUCCES	This driver is added to ControllerHandle.
EFI_ALREADY_STARTED	This driver is already running on ControllerHandle.
other	This driver does not support this device.

References _IP6_SERVICE::FasterTimer, IP6_ONE_SECOND_IN_MS, IP6_TIMER_INTERVAL_IN_MS, Ip6AcceptFrame(), Ip6CleanService(), _IP6_CONFIG_INSTANCE::Ip6Config, _IP6_SERVICE::Ip6ConfigInstance, Ip6CreateService(), Ip6ReceiveFrame(), mIp6Id, _IP6_SERVICE::ServiceBinding, and _IP6_SERVICE::Timer.

```
EFI_STATUS EFIAPI
Ip6DriverBindingStop (
    IN EFI_DRIVER_BINDING_PROTOCOL *  This,
    IN EFI_HANDLE                     ControllerHandle,
    IN UINTN                          NumberOfChildren,
    IN EFI_HANDLE *                   ChildHandleBuffer
)
```

Stop this driver on ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to stop driver on.
[in]	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If number of children is zero, stop the entire bus driver.
[in]	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References IP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer, _IP6_SERVICE::Children, _IP6_CONFIG_INSTANCE::Dhcp6Event, IP6_SERVICE_DESTROY, IP6_SERVICE_FROM_PROTOCOL, Ip6CleanService(), _IP6_CONFIG_INSTANCE::Ip6Config, Ip6ConfigDestroyDhcp6(), _IP6_SERVICE::Ip6ConfigInstance, Ip6DestroyChildEntryInHandleBuffer(), IP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren, IP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding, and _IP6_SERVICE::State.

```
EFI_STATUS EFIAPI
Ip6DriverBindingSupported (
    IN EFI_DRIVER_BINDING_PROTOCOL *  This,
    IN EFI_HANDLE                     ControllerHandle,
    IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
)
```

Test to see if this driver supports ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to test.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	This driver supports this device.
EFI_ALREADY_STARTED	This driver is already running on this device.
other	This driver does not support this device.

EFI_STATUS EFIAPI Ip6DriverEntryPoint (IN EFI_HANDLE *ImageHandle*, IN EFI_SYSTEM_TABLE * *SystemTable*)

This is the declaration of an EFI image entry point. This entry point is the same for UEFI Applications, UEFI OS Loaders, and UEFI Drivers including both device drivers and bus drivers.

The entry point for IP6 driver which installs the driver binding and component name protocol on its image.

Parameters:

[in]	*ImageHandle*	The firmware allocated handle for the UEFI image.
[in]	*SystemTable*	A pointer to the EFI System Table.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.

References gIp6ComponentName, gIp6ComponentName2, gIp6DriverBinding, and IpSec2InstalledCallback().

EFI_STATUS EFIAPI Ip6ServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL * *This*, IN EFI_HANDLE * *ChildHandle*)

Creates a child handle with a set of I/O services.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ChildHandle*	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is not NULL, then the I/O services are added to the existing child handle.

Return values:

EFI_SUCCES	The child handle was created with the I/O services.
EFI_OUT_OF_RESOURCES	There are not enough resources availabe to create the child.
other	The child handle was not created.

References _IP6_SERVICE::Children, gIp6DriverBinding, _IP6_PROTOCOL::Handle, IP6_SERVICE_FROM_PROTOCOL, Ip6CleanProtocol(), Ip6InitProtocol(), _IP6_PROTOCOL::Ip6Proto, _IP6_PROTOCOL::Link, _IP6_SERVICE::LinkLocalDadFail, _IP6_SERVICE::MnpChildHandle, and _IP6_SERVICE::NumChildren.

Referenced by Ip6CreateService().

EFI_STATUS EFIAPI Ip6ServiceBindingDestroyChild (IN EFI_SERVICE_BINDING_PROTOCOL * *This*, IN EFI_HANDLE *ChildHandle*)

Destroys a child handle with a set of I/O services.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ChildHandle*	Handle of the child to destroy.

Return values:

EFI_SUCCES	The I/O services were removed from the child handle.
EFI_UNSUPPORTED	The child handle does not support the I/O services that are being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
EFI_ACCESS_DENIED	The child handle could not be destroyed because its I/O services are being used.
other	The child handle was not destroyed.

References gIp6DriverBinding, _IP6_PROTOCOL::InDestroy, IP6_INSTANCE_FROM_PROTOCOL, IP6_SERVICE_FROM_PROTOCOL, Ip6CleanProtocol(), _IP6_PROTOCOL::Ip6Proto, _IP6_PROTOCOL::Link, _IP6_SERVICE::MnpChildHandle, _IP6_SERVICE::NumChildren, and _IP6_PROTOCOL::Service.

Referenced by Ip6CreateService().

VOID EFIAPI IpSec2InstalledCallback (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

Callback function for IpSec2 Protocol install.

Parameters:

[in]	*Event*	Event whose notification function is being invoked
[in]	*Context*	Pointer to the notification function's context

References mIpSec2Installed.

Referenced by Ip6DriverEntryPoint().

Variable Documentation

EFI_DRIVER_BINDING_PROTOCOL gIp6DriverBinding

Initial value:

```
{
 Ip6DriverBindingSupported,
 Ip6DriverBindingStart,
 Ip6DriverBindingStop,
 0xa,
 NULL,
 NULL
}
```

Referenced by Ip6DriverEntryPoint(), Ip6ServiceBindingCreateChild(), and Ip6ServiceBindingDestroyChild().

BOOLEAN mIpSec2Installed = FALSE

Referenced by Ip6IpSecProcessPacket(), and IpSec2InstalledCallback().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Ip6Dxe/Ip6Driver.h File Reference

Data Structures

struct IP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT

Functions

EFI_STATUS Ip6CleanService (IN IP6_SERVICE *IpSb)

EFI_STATUS EFIAPI Ip6DriverEntryPoint (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)

EFI_STATUS EFIAPI Ip6DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI Ip6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI Ip6DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)

EFI_STATUS EFIAPI Ip6ServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE *ChildHandle)

EFI_STATUS EFIAPI Ip6ServiceBindingDestroyChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE ChildHandle)

Variables

EFI_DRIVER_BINDING_PROTOCOL gIp6DriverBinding

EFI_COMPONENT_NAME_PROTOCOL gIp6ComponentName

EFI_COMPONENT_NAME2_PROTOCOL gIp6ComponentName2

EFI_UNICODE_STRING_TABLE * gIp6ControllerNameTable

Detailed Description

The driver binding and service binding protocol for IP6 driver.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS Ip6CleanService (IN IP6_SERVICE * *IpSb*)

Clean up an IP6 service binding instance. It releases all the resource allocated by the instance. The instance may be partly initialized, or partly destroyed. If a resource is destroyed, it is marked as that in case the destroy failed and being called again later.

Parameters:

`[in]` *IpSb* The IP6 service binding instance to clean up.

Return values:

EFI_SUCCESS The resource used by the instance are cleaned up.
Others Failed to clean up some of the resources.

References IP6_LINK_LOCAL_SCOPE, Ip6CleanAssembleTable(), Ip6CleanDefaultRouterList(), Ip6CleanInterface(), Ip6CleanPrefixListTable(), Ip6CleanRouteTable(), Ip6ConfigCleanInstance(), Ip6FreeNeighborEntry(), Ip6LeaveGroup(), and Ip6SetToAllNodeMulticast().

Referenced by Ip6CreateService(), Ip6DriverBindingStart(), and Ip6DriverBindingStop().

EFI_STATUS EFIAPI Ip6DriverBindingStart
(IN EFI_DRIVER_BINDING_PROTOCOL * *This*,
IN EFI_HANDLE *ControllerHandle*,
IN EFI_DEVICE_PATH_PROTOCOL * *RemainingDevicePath*
)

Start this driver on ControllerHandle.

Parameters:

`[in]` *This* Protocol instance pointer.
`[in]` *ControllerHandle* Handle of device to bind driver to.
`[in]` *RemainingDevicePath* Optional parameter used to pick a specific child device to start.

Return values:

EFI_SUCCES This driver is added to ControllerHandle.
EFI_ALREADY_STARTED This driver is already running on ControllerHandle.
other This driver does not support this device.

References _IP6_SERVICE::FasterTimer, IP6_ONE_SECOND_IN_MS, IP6_TIMER_INTERVAL_IN_MS, Ip6AcceptFrame(), Ip6CleanService(), _IP6_CONFIG_INSTANCE::Ip6Config, _IP6_SERVICE::Ip6ConfigInstance, Ip6CreateService(), Ip6ReceiveFrame(), mIp6Id, _IP6_SERVICE::ServiceBinding, and _IP6_SERVICE::Timer.

EFI_STATUS EFIAPI Ip6DriverBindingStop
(IN EFI_DRIVER_BINDING_PROTOCOL * *This*,
IN EFI_HANDLE *ControllerHandle*,
IN UINTN *NumberOfChildren*,
IN EFI_HANDLE * *ChildHandleBuffer*

)

Stop this driver on ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to stop driver on.
[in]	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If number of children is zero, stop the entire bus driver.
[in]	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References IP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer, _IP6_SERVICE::Children, _IP6_CONFIG_INSTANCE::Dhcp6Event, IP6_SERVICE_DESTROY, IP6_SERVICE_FROM_PROTOCOL, Ip6CleanService(), _IP6_CONFIG_INSTANCE::Ip6Config, Ip6ConfigDestroyDhcp6(), _IP6_SERVICE::Ip6ConfigInstance, Ip6DestroyChildEntryInHandleBuffer(), IP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren, IP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding, and _IP6_SERVICE::State.

```
EFI_STATUS EFIAPI
Ip6DriverBindingSupported ( IN EFI_DRIVER_BINDING_PROTOCOL * This,
                            IN EFI_HANDLE ControllerHandle,
                            IN EFI_DEVICE_PATH_PROTOCOL * RemainingDevicePath
                            )
```

Test to see if this driver supports ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to test.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	This driver supports this device.
EFI_ALREADY_STARTED	This driver is already running on this device.
other	This driver does not support this device.

```
EFI_STATUS EFIAPI Ip6DriverEntryPoint ( IN EFI_HANDLE ImageHandle,
                                        IN EFI_SYSTEM_TABLE * SystemTable
                                        )
```

This is the declaration of an EFI image entry point. This entry point is the same for UEFI Applications, UEFI OS Loaders, and UEFI Drivers including both device drivers and bus drivers.

The entry point for IP6 driver which installs the driver binding and component name protocol on its image.

Parameters:

[in]	*ImageHandle*	The firmware allocated handle for the UEFI image.
[in]	*SystemTable*	A pointer to the EFI System Table.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.

References gIp6ComponentName, gIp6ComponentName2, gIp6DriverBinding, and IpSec2InstalledCallback().

```
EFI_STATUS EFIAPI
Ip6ServiceBindingCreateChild ( IN EFI_SERVICE_BINDING_PROTOCOL * This,
                               IN EFI_HANDLE * ChildHandle )
```

Creates a child handle with a set of I/O services.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ChildHandle*	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is not NULL, then the I/O services are added to the existing child handle.

Return values:

EFI_SUCCES	The child handle was created with the I/O services.
EFI_OUT_OF_RESOURCES	There are not enough resources availabe to create the child.
other	The child handle was not created.

References _IP6_SERVICE::Children, gIp6DriverBinding, _IP6_PROTOCOL::Handle, IP6_SERVICE_FROM_PROTOCOL, Ip6CleanProtocol(), Ip6InitProtocol(), _IP6_PROTOCOL::Ip6Proto, _IP6_PROTOCOL::Link, _IP6_SERVICE::LinkLocalDadFail, _IP6_SERVICE::MnpChildHandle, and _IP6_SERVICE::NumChildren.

Referenced by Ip6CreateService().

```
EFI_STATUS EFIAPI
Ip6ServiceBindingDestroyChild ( IN EFI_SERVICE_BINDING_PROTOCOL * This,
                                IN EFI_HANDLE ChildHandle )
```

Destroys a child handle with a set of I/O services.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ChildHandle*	Handle of the child to destroy.

Return values:

EFI_SUCCES	The I/O services were removed from the child handle.
EFI_UNSUPPORTED	The child handle does not support the I/O services that are being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
EFI_ACCESS_DENIED	The child handle could not be destroyed because its I/O services are being used.
other	The child handle was not destroyed.

Destroys a child handle with a set of I/O services.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ChildHandle*	Handle of the child to destroy.

Return values:

EFI_SUCCES	The I/O services were removed from the child handle.
EFI_UNSUPPORTED	The child handle does not support the I/O services that are being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
EFI_ACCESS_DENIED	The child handle could not be destroyed because its I/O services are being used.
other	The child handle was not destroyed.

References gIp6DriverBinding, _IP6_PROTOCOL::InDestroy, IP6_INSTANCE_FROM_PROTOCOL, IP6_SERVICE_FROM_PROTOCOL, Ip6CleanProtocol(), _IP6_PROTOCOL::Ip6Proto, _IP6_PROTOCOL::Link, _IP6_SERVICE::MnpChildHandle, _IP6_SERVICE::NumChildren, and _IP6_PROTOCOL::Service.

Referenced by Ip6CreateService().

Variable Documentation

EFI_COMPONENT_NAME_PROTOCOL gIp6ComponentName

Referenced by Ip6ComponentNameGetControllerName(), Ip6ComponentNameGetDriverName(), Ip6DriverEntryPoint(), and UpdateName().

EFI_COMPONENT_NAME2_PROTOCOL gIp6ComponentName2

Referenced by Ip6DriverEntryPoint(), and UpdateName().

EFI_UNICODE_STRING_TABLE* gIp6ControllerNameTable

Referenced by Ip6ComponentNameGetControllerName(), and UpdateName().

EFI_DRIVER_BINDING_PROTOCOL gIp6DriverBinding

Referenced by Ip6DriverEntryPoint(), Ip6ServiceBindingCreateChild(), and Ip6ServiceBindingDestroyChild().

- Globals

NetworkPkg/Ip6Dxe/Ip6Icmp.c File Reference

Functions

EFI_STATUS Ip6IcmpReplyEcho (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
EFI_STATUS Ip6ProcessPacketTooBig (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
EFI_STATUS Ip6ProcessIcmpError (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
EFI_STATUS Ip6ProcessIcmpInformation (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
EFI_STATUS Ip6IcmpHandle (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
VOID Ip6GetPrefix (IN UINT8 PrefixLength, IN OUT EFI_IPv6_ADDRESS *Prefix)
BOOLEAN Ip6IsAnycast (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *DestinationAddress)
EFI_STATUS Ip6SendIcmpError (IN IP6_SERVICE *IpSb, IN NET_BUF *Packet, IN EFI_IPv6_ADDRESS *SourceAddress, IN EFI_IPv6_ADDRESS *DestinationAddress, IN UINT8 Type, IN UINT8 Code, IN UINT32 *Pointer)

Variables

EFI_IP6_ICMP_TYPE mIp6SupportedIcmp [23]

Detailed Description

The ICMPv6 handle routines to process the ICMPv6 control messages.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
VOID Ip6GetPrefix ( IN UINT8                    PrefixLength,
                    IN OUT EFI_IPv6_ADDRESS *   Prefix
                  )
```

Retrieve the Prefix address according to the PrefixLength by clear the useless bits.

Parameters:

`[in]` *PrefixLength* The prefix length of the prefix.

[in,out] *Prefix* On input, points to the original prefix address with dirty bits; on output, points to the updated address with useless bit clear.

Referenced by Ip6IsAnycast().

```
EFI_STATUS Ip6IcmpHandle ( IN IP6_SERVICE *        IpSb,
                           IN EFI_IP6_HEADER *     Head,
                           IN NET_BUF *            Packet
                         )
```

Handle the ICMPv6 packet. First validate the message format, then, according to the message types, process it as an informational packet or an error packet.

Parameters:

[in] *IpSb* The IP service that received the packet.
[in] *Head* The IP head of the ICMPv6 packet.
[in] *Packet* The content of the ICMPv6 packet with IP head removed.

Return values:

EFI_INVALID_PARAMETER The packet is malformated.
EFI_SUCCESS The ICMPv6 message successfully processed.
Others Failed to handle the ICMPv6 packet.

References ICMP_V6_ERROR_MAX, Ip6ProcessIcmpError(), and Ip6ProcessIcmpInformation().

Referenced by Ip6AcceptFrame().

```
EFI_STATUS Ip6IcmpReplyEcho ( IN IP6_SERVICE *        IpSb,
                              IN EFI_IP6_HEADER *     Head,
                              IN NET_BUF *            Packet
                            )
```

Reply an ICMPv6 echo request.

Parameters:

[in] *IpSb* The IP service that received the packet.
[in] *Head* The IP head of the ICMPv6 informational message.
[in] *Packet* The content of the ICMPv6 message with the IP head removed.

Return values:

EFI_OUT_OF_RESOURCES Failed to allocate resources.
EFI_SUCCESS Successfully answered the ICMPv6 Echo request.
Others Failed to answer the ICMPv6 Echo request.

References IP6_MAX_HEADLEN, Ip6IsOneOfSetAddress(), Ip6Output(), and Ip6SysPacketSent().

Referenced by Ip6ProcessIcmpInformation().

```
BOOLEAN Ip6IsAnycast ( IN IP6_SERVICE *        IpSb,
                       IN EFI_IPv6_ADDRESS *   DestinationAddress
```

)

Check whether the DestinationAddress is an anycast address.

Parameters:

[in]	*IpSb*	The IP service that received the packet.
[in]	*DestinationAddress*	Points to the Destination Address of the packet.

Return values:

TRUE	The DestinationAddress is anycast address.
FALSE	The DestinationAddress is not anycast address.

References Ip6FindPrefixListEntry(), Ip6GetPrefix(), _IP6_PREFIX_LIST_ENTRY::Prefix, and _IP6_PREFIX_LIST_ENTRY::PrefixLength.

Referenced by Ip6Output(), and Ip6SendIcmpError().

```
EFI_STATUS Ip6ProcessIcmpError ( IN IP6_SERVICE *        IpSb,
                                 IN EFI_IP6_HEADER *     Head,
                                 IN NET_BUF *            Packet
                               )
```

Process the ICMPv6 error packet, and deliver the packet to upper layer.

Parameters:

[in]	*IpSb*	The IP service that received the packet.
[in]	*Head*	The IP head of the ICMPv6 error packet.
[in]	*Packet*	The content of the ICMPv6 error with the IP head removed.

Return values:

EFI_SUCCESS	The ICMPv6 error processed successfully.
EFI_INVALID_PARAMETER	The packet is invalid.
Others	Failed to process the packet.

References IP6_GET_CLIP_INFO, Ip6Demultiplex(), and Ip6ProcessPacketTooBig().

Referenced by Ip6IcmpHandle().

```
EFI_STATUS Ip6ProcessIcmpInformation ( IN IP6_SERVICE *        IpSb,
                                       IN EFI_IP6_HEADER *     Head,
                                       IN NET_BUF *            Packet
                                     )
```

Process the ICMPv6 informational messages. If it is an ICMPv6 echo request, answer it. If it is a MLD message, trigger MLD routines to process it. If it is a ND message, trigger ND routines to process it. Otherwise, deliver it to upper layer.

Parameters:

[in]	*IpSb*	The IP service that receivd the packet.
[in]	*Head*	The IP head of the ICMPv6 informational packet.
[in]	*Packet*	The content of the ICMPv6 informational packet with IP head removed.

Return values:

EFI_INVALID_PARAMETER	The packet is invalid.
EFI_SUCCESS	The ICMPv6 informational message processed.
Others	Failed to process ICMPv6 informational message.

References IP6_SERVICE_SIGNATURE, Ip6Demultiplex(), Ip6IcmpReplyEcho(), Ip6ProcessMldQuery(), Ip6ProcessMldReport(), Ip6ProcessNeighborAdvertise(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), and Ip6ProcessRouterAdvertise().

Referenced by Ip6IcmpHandle().

```
EFI_STATUS Ip6ProcessPacketTooBig ( IN IP6_SERVICE *        IpSb,
                                    IN EFI_IP6_HEADER *     Head,
                                    IN NET_BUF *            Packet
                                  )
```

Process Packet Too Big message sent by a router in response to a packet that it cannot forward because the packet is larger than the MTU of outgoing link. Since this driver already uses IPv6 minimum link MTU as the maximum packet size, if Packet Too Big message is still received, do not reduce the packet size, but rather include a Fragment header in the subsequent packets.

Parameters:

[in]	*IpSb*	The IP service that received the packet.
[in]	*Head*	The IP head of the ICMPv6 error packet.
[in]	*Packet*	The content of the ICMPv6 error with the IP head removed.

Return values:

EFI_SUCCESS	The ICMPv6 error processed successfully.
EFI_OUT_OF_RESOURCES	Failed to finish the operation due to lack of resource.
EFI_NOT_FOUND	The packet too big message is not sent to us.

References IP6_ROUTE_ENTRY::Flag, IP6_DIRECT_ROUTE, IP6_MIN_LINK_MTU, IP6_PACKET_TOO_BIG, Ip6CreateRouteEntry(), Ip6FindRouteEntry(), Ip6FreeRouteEntry(), and IP6_ROUTE_ENTRY::Link.

Referenced by Ip6ProcessIcmpError().

```
EFI_STATUS Ip6SendIcmpError ( IN IP6_SERVICE *          IpSb,
                              IN NET_BUF *              Packet,
                              IN EFI_IPv6_ADDRESS *     SourceAddress,
                              IN EFI_IPv6_ADDRESS *     DestinationAddress,
                              IN UINT8                  Type,
                              IN UINT8                  Code,
                              IN UINT32 *               Pointer
                            )
```

Generate ICMPv6 error message and send it out to DestinationAddress. Currently Destination Unreachable message, Time Exceeded message and Parameter Problem message are supported.

Parameters:

[in]	*IpSb*	The IP service that received the packet.
[in]	*Packet*	The packet which invoking ICMPv6 error.

[in]	*SourceAddress*	If not NULL, points to the SourceAddress. Otherwise, the IP layer will select a source address according to the DestinationAddress.
[in]	*DestinationAddress*	Points to the Destination Address of the ICMPv6 error message.
[in]	*Type*	The type of the ICMPv6 message.
[in]	*Code*	The additional level of the ICMPv6 message.
[in]	*Pointer*	If not NULL, identifies the octet offset within the invoking packet where the error was detected.

Return values:

EFI_INVALID_PARAMETER	The packet is malformated.
EFI_OUT_OF_RESOURCES	There is no sufficient resource to complete the operation.
EFI_SUCCESS	The ICMPv6 message was successfully sent out.
Others	Failed to generate the ICMPv6 packet.

References Ip6IsAnycast(), Ip6Output(), and Ip6SysPacketSent().

Referenced by Ip6FreeNeighborEntry(), Ip6IsExtsValid(), Ip6IsOptionValid(), and Ip6PacketTimerTicking().

Variable Documentation

EFI_IP6_ICMP_TYPE mIp6SupportedIcmp[23]

Referenced by EfiIp6GetModeData().

NetworkPkg/Ip6Dxe/Ip6Icmp.h File Reference

Defines

"memItemRight" valign="bottom">ICMP_V6_DEFAULT_CODE 0

#define ICMP_V6_ERROR_MAX 127
#define ICMP_V6_INVALID_MESSAGE 0
#define ICMP_V6_ERROR_MESSAGE 1
#define ICMP_V6_INFORMATION_MESSAGE 2

Functions

EFI_STATUS Ip6IcmpHandle (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
BOOLEAN Ip6IsAnycast (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *DestinationAddress)
EFI_STATUS Ip6SendIcmpError (IN IP6_SERVICE *IpSb, IN NET_BUF *Packet, IN EFI_IPv6_ADDRESS *SourceAddress, IN EFI_IPv6_ADDRESS *DestinationAddress, IN UINT8 Type, IN UINT8 Code, IN UINT32 *Pointer)

Variables

EFI_IP6_ICMP_TYPE mIp6SupportedIcmp []

Detailed Description

Header file for ICMPv6 protocol.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

#define ICMP_V6_ERROR_MAX 127

Referenced by Ip6IcmpHandle(), and Ip6InstanceFrameAcceptable().

#define ICMP_V6_ERROR_MESSAGE 1

#define ICMP_V6_INFORMATION_MESSAGE 2

#define ICMP_V6_INVALID_MESSAGE 0

Function Documentation

```
EFI_STATUS Ip6IcmpHandle ( IN IP6_SERVICE *        IpSb,
                           IN EFI_IP6_HEADER *     Head,
                           IN NET_BUF *            Packet
                         )
```

Handle the ICMPv6 packet. First validate the message format, then, according to the message types, process it as an informational packet or an error packet.

Parameters:

[in]	*IpSb*	The IP service that received the packet.
[in]	*Head*	The IP head of the ICMPv6 packet.
[in]	*Packet*	The content of the ICMPv6 packet with IP head removed.

Return values:

EFI_INVALID_PARAMETER	The packet is malformated.
EFI_SUCCESS	The ICMPv6 message successfully processed.
Others	Failed to handle the ICMPv6 packet.

References ICMP_V6_ERROR_MAX, Ip6ProcessIcmpError(), and Ip6ProcessIcmpInformation().

Referenced by Ip6AcceptFrame().

```
BOOLEAN Ip6IsAnycast ( IN IP6_SERVICE *        IpSb,
                       IN EFI_IPv6_ADDRESS *   DestinationAddress
                     )
```

Check whether the DestinationAddress is an anycast address.

Parameters:

[in]	*IpSb*	The IP service that received the packet.
[in]	*DestinationAddress*	Points to the Destination Address of the packet.

Return values:

TRUE	The DestinationAddress is anycast address.
FALSE	The DestinationAddress is not anycast address.

References Ip6FindPrefixListEntry(), Ip6GetPrefix(), _IP6_PREFIX_LIST_ENTRY::Prefix, and _IP6_PREFIX_LIST_ENTRY::PrefixLength.

Referenced by Ip6Output(), and Ip6SendIcmpError().

```
EFI_STATUS Ip6SendIcmpError ( IN IP6_SERVICE *          IpSb,
                              IN NET_BUF *              Packet,
                              IN EFI_IPv6_ADDRESS *     SourceAddress,
                              IN EFI_IPv6_ADDRESS *     DestinationAddress,
                              IN UINT8                  Type,
                              IN UINT8                  Code,
                              IN UINT32 *               Pointer
                            )
```

Generate ICMPv6 error message and send it out to DestinationAddress. Currently Destination Unreachable message, Time Exceeded message and Parameter Problem message are supported.

Parameters:

[in]	*IpSb*	The IP service that received the packet.
[in]	*Packet*	The packet which invoking ICMPv6 error.
[in]	*SourceAddress*	If not NULL, points to the SourceAddress. Otherwise, the IP layer will select a source address according to the DestinationAddress.
[in]	*DestinationAddress*	Points to the Destination Address of the ICMPv6 error message.
[in]	*Type*	The type of the ICMPv6 message.
[in]	*Code*	The additional level of the ICMPv6 message.
[in]	*Pointer*	If not NULL, identifies the octet offset within the invoking packet where the error was detected.

Return values:

EFI_INVALID_PARAMETER	The packet is malformated.
EFI_OUT_OF_RESOURCES	There is no sufficient resource to complete the operation.
EFI_SUCCESS	The ICMPv6 message was successfully sent out.
Others	Failed to generate the ICMPv6 packet.

References Ip6IsAnycast(), Ip6Output(), and Ip6SysPacketSent().

Referenced by Ip6FreeNeighborEntry(), Ip6IsExtsValid(), Ip6IsOptionValid(), and Ip6PacketTimerTicking().

Variable Documentation

EFI_IP6_ICMP_TYPE mIp6SupportedIcmp[]

Referenced by EfiIp6GetModeData().

NetworkPkg/Ip6Dxe/Ip6If.c File Reference

Functions

VOID EFIAPI Ip6OnFrameSent (IN EFI_EVENT Event, IN VOID *Context)
BOOLEAN Ip6CancelInstanceFrame (IN IP6_LINK_TX_TOKEN *Frame, IN VOID *Context)
EFI_STATUS Ip6SetAddress (IN IP6_INTERFACE *Interface, IN EFI_IPv6_ADDRESS *Ip6Addr, IN BOOLEAN IsAnycast, IN UINT8 PrefixLength, IN UINT32 ValidLifetime, IN UINT32 PreferredLifetime, IN IP6_DAD_CALLBACK DadCallback, IN VOID *Context)
IP6_INTERFACE * Ip6CreateInterface (IN IP6_SERVICE *IpSb, IN BOOLEAN LinkLocal)
VOID Ip6CleanInterface (IN IP6_INTERFACE *Interface, IN IP6_PROTOCOL *IpInstance)
IP6_LINK_TX_TOKEN * Ip6CreateLinkTxToken (IN IP6_INTERFACE *Interface, IN IP6_PROTOCOL *IpInstance, IN NET_BUF *Packet, IN IP6_FRAME_CALLBACK CallBack, IN VOID *Context)
VOID Ip6FreeLinkTxToken (IN IP6_LINK_TX_TOKEN *Token)
VOID EFIAPI Ip6RecycleFrame (IN VOID *Context)
VOID EFIAPI Ip6OnFrameReceivedDpc (IN VOID *Context)
VOID EFIAPI Ip6OnFrameReceived (IN EFI_EVENT Event, IN VOID *Context)
EFI_STATUS Ip6ReceiveFrame (IN IP6_FRAME_CALLBACK CallBack, IN IP6_SERVICE *IpSb)
VOID EFIAPI Ip6OnFrameSentDpc (IN VOID *Context)
EFI_STATUS Ip6SendFrame (IN IP6_INTERFACE *Interface, IN IP6_PROTOCOL *IpInstance, IN NET_BUF *Packet, IN EFI_IPv6_ADDRESS *NextHop, IN IP6_FRAME_CALLBACK CallBack, IN VOID *Context)
VOID EFIAPI Ip6TimerTicking (IN EFI_EVENT Event, IN VOID *Context)

Detailed Description

Implement IP6 pesudo interface.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
BOOLEAN Ip6CancelInstanceFrame ( IN IP6_LINK_TX_TOKEN *   Frame,
                                 IN VOID *                Context
                               )
```

Fileter function to cancel all the frame related to an IP instance.

Parameters:

[in] *Frame* The transmit request to test whether to cancel.

[in] *Context* The context which is the Ip instance that issued the transmit.

Return values:

TRUE The frame belongs to this instance and is to be removed.

FALSE The frame doesn't belong to this instance.

Referenced by Ip6CleanInterface().

```
VOID Ip6CleanInterface ( IN IP6_INTERFACE *   Interface,
                         IN IP6_PROTOCOL *    IpInstance
                       )
```

Free the interface used by IpInstance. All the IP instance with the same Ip/prefix pair share the same interface. It is reference counted. All the frames that haven't been sent will be cancelled. Because the IpInstance is optional, the caller must remove IpInstance from the interface's instance list.

Parameters:

[in] *Interface* The interface used by the IpInstance.

[in] *IpInstance* The IP instance that free the interface. NULL if the IP driver is releasing the default interface.

References IP6_INTERFACE_SIGNATURE, Ip6CancelFrames(), Ip6CancelInstanceFrame(), and Ip6RemoveAddr().

Referenced by Ip6CleanProtocol(), and Ip6CleanService().

```
IP6_INTERFACE* Ip6CreateInterface ( IN IP6_SERVICE *   IpSb,
                                    IN BOOLEAN         LinkLocal
                                  )
```

Create an IP6_INTERFACE.

Parameters:

[in] *IpSb* The IP6 service binding instance.

[in] *LinkLocal* If TRUE, the instance is created for link-local address. Otherwise, it is not for a link-local address.

Returns:

Point to the created IP6_INTERFACE, otherwise NULL.

References _IP6_INTERFACE::AddressCount, _IP6_INTERFACE::AddressList, _IP6_INTERFACE::ArpQues, _IP6_INTERFACE::Configured, _IP6_INTERFACE::Controller, _IP6_INTERFACE::DelayJoinList, _IP6_INTERFACE::DupAddrDetect, _IP6_INTERFACE::DupAddrDetectList, _IP6_INTERFACE::Image, IP6_INFINIT_LIFETIME, IP6_INTERFACE_SIGNATURE, IP6_LINK_LOCAL_PREFIX_LENGTH, IP6_SERVICE_SIGNATURE, Ip6CreateLinkLocalAddr(), Ip6SetAddress(), _IP6_INTERFACE::IpInstances, _IP6_INTERFACE::PromiscRecv, _IP6_INTERFACE::RefCnt, _IP6_INTERFACE::SentFrames, _IP6_INTERFACE::Service, and _IP6_INTERFACE::Signature.

Referenced by Ip6CreateService().

```
IP6_LINK_TX_TOKEN* Ip6CreateLinkTxToken ( IN IP6_INTERFACE *          Interface,
                                          IN IP6_PROTOCOL *           IpInstance,
                                          IN NET_BUF *                Packet,
                                          IN IP6_FRAME_CALLBACK       CallBack,
                                          IN VOID *                   Context
                                          )
```

Create and wrap a transmit request into a newly allocated IP6_LINK_TX_TOKEN.

Parameters:

[in]	*Interface*	The interface to send out from.
[in]	*IpInstance*	The IpInstance that transmit the packet. NULL if the packet is sent by the IP6 driver itself.
[in]	*Packet*	The packet to transmit
[in]	*CallBack*	Call back function to execute if transmission finished.
[in]	*Context*	Opaque parameter to the callback.

Returns:

The wrapped token if succeed or NULL.

References IP6_LINK_TX_TOKEN::CallBack, IP6_LINK_TX_TOKEN::Context, IP6_LINK_TX_TOKEN::DstMac, IP6_ETHER_PROTO, IP6_LINK_TX_SIGNATURE, Ip6OnFrameSent(), IP6_LINK_TX_TOKEN::IpInstance, IP6_LINK_TX_TOKEN::Link, IP6_LINK_TX_TOKEN::MnpToken, IP6_LINK_TX_TOKEN::MnpTxData, IP6_LINK_TX_TOKEN::Packet, IP6_LINK_TX_TOKEN::Signature, and IP6_LINK_TX_TOKEN::SrcMac.

Referenced by Ip6SendFrame().

```
VOID Ip6FreeLinkTxToken ( IN IP6_LINK_TX_TOKEN *   Token   )
```

Free the link layer transmit token. It will close the event, then free the memory used.

Parameters:

[in]	*Token*	Token to free.

References IP6_LINK_TX_SIGNATURE.

Referenced by Ip6FreeNeighborEntry(), Ip6OnArpResolved(), Ip6OnFrameSentDpc(), and Ip6SendFrame().

```
VOID EFIAPI Ip6OnFrameReceived ( IN EFI_EVENT   Event,
                                 IN VOID *      Context
                                 )
```

Request Ip6OnFrameReceivedDpc as a DPC at TPL_CALLBACK.

Parameters:

Event	The receive event delivered to MNP for receive.
Context	Context for the callback.

References Ip6OnFrameReceivedDpc().

Referenced by Ip6CreateService().

VOID EFIAPI Ip6OnFrameReceivedDpc (IN VOID * *Context*)

Received a frame from MNP. Wrap it in net buffer then deliver it to IP's input function. The ownship of the packet also is transferred to IP. When Ip is finished with this packet, it will call NetbufFree to release the packet, NetbufFree will again call the Ip6RecycleFrame to signal MNP's event and free the token used.

Parameters:

`[in]` *Context* Context for the callback.

References IP6_LINK_RX_TOKEN::CallBack, IP6_LINK_RX_TOKEN::Context, IP6_LINK_BROADCAST, IP6_LINK_MULTICAST, IP6_LINK_PROMISC, IP6_LINK_RX_SIGNATURE, IP6_MAX_HEADLEN, IP6_SERVICE_SIGNATURE, Ip6RecycleFrame(), and IP6_LINK_RX_TOKEN::MnpToken.

Referenced by Ip6OnFrameReceived().

VOID EFIAPI Ip6OnFrameSent (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

Request Ip6OnFrameSentDpc as a DPC at TPL_CALLBACK.

Parameters:

`[in]` *Event* The transmit token's event.

`[in]` *Context* The Context which is pointed to the token.

Request Ip6OnFrameSentDpc as a DPC at TPL_CALLBACK.

Parameters:

`[in]` *Event* The transmit token's event.

`[in]` *Context* Context which points to the token.

References Ip6OnFrameSentDpc().

Referenced by Ip6CreateLinkTxToken().

VOID EFIAPI Ip6OnFrameSentDpc (IN VOID * *Context*)

Callback funtion when frame transmission is finished. It will call the frame owner's callback function to tell it the result.

Parameters:

`[in]` *Context* Context which points to the token.

References IP6_LINK_TX_TOKEN::CallBack, IP6_LINK_TX_TOKEN::Context, IP6_LINK_TX_SIGNATURE, Ip6FreeLinkTxToken(), IP6_LINK_TX_TOKEN::Link, IP6_LINK_TX_TOKEN::MnpToken, and IP6_LINK_TX_TOKEN::Packet.

Referenced by Ip6OnFrameSent().

EFI_STATUS Ip6ReceiveFrame (IN IP6_FRAME_CALLBACK *CallBack*,
IN IP6_SERVICE * *IpSb*

)

Request to receive the packet from the interface.

Parameters:

[in]	*CallBack*	Function to call when receive finished.
[in]	*IpSb*	Points to IP6 service binding instance.

Return values:

EFI_ALREADY_STARTED	There is already a pending receive request.
EFI_OUT_OF_RESOURCES	Failed to allocate resource to receive.
EFI_SUCCESS	The recieve request has been started.

References IP6_LINK_RX_TOKEN::CallBack, IP6_LINK_RX_TOKEN::Context, IP6_SERVICE_SIGNATURE, and IP6_LINK_RX_TOKEN::MnpToken.

Referenced by Ip6AcceptFrame(), and Ip6DriverBindingStart().

VOID EFIAPI Ip6RecycleFrame (IN VOID * *Context*)

Callback function when the received packet is freed. Check Ip6OnFrameReceived for information.

Parameters:

[in]	*Context*	Points to EFI_MANAGED_NETWORK_RECEIVE_DATA.

Referenced by Ip6OnFrameReceivedDpc().

```
EFI_STATUS Ip6SendFrame ( IN IP6_INTERFACE *          Interface,
                          IN IP6_PROTOCOL *           IpInstance,
                          IN NET_BUF *                Packet,
                          IN EFI_IPv6_ADDRESS *       NextHop,
                          IN IP6_FRAME_CALLBACK       CallBack,
                          IN VOID *                   Context
                        )
```

Send a frame from the interface. If the next hop is a multicast address, it is transmitted immediately. If the next hop is a unicast, and the NextHop's MAC is not known, it will perform address resolution. If an error occurred, the CallBack won't be called. So, the caller must test the return value, and take action when there is an error.

Parameters:

[in]	*Interface*	The interface to send the frame from
[in]	*IpInstance*	The IP child that request the transmission. NULL if it is the IP6 driver itself.
[in]	*Packet*	The packet to transmit.
[in]	*NextHop*	The immediate destination to transmit the packet to.
[in]	*CallBack*	Function to call back when transmit finished.
[in]	*Context*	Opaque parameter to the callback.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate resource to send the frame.
EFI_NO_MAPPING	Can't resolve the MAC for the nexthop.
EFI_SUCCESS	The packet successfully transmitted.

References _IP6_NEIGHBOR_ENTRY::ArpFree, _IP6_NEIGHBOR_ENTRY::ArpList, IP6_LINK_TX_TOKEN::DstMac, _IP6_NEIGHBOR_ENTRY::Frames, _IP6_NEIGHBOR_ENTRY::Interface, IP6_DELAY_FIRST_PROBE_TIME, IP6_GET_TICKS, IP6_SERVICE_SIGNATURE, Ip6CreateLinkTxToken(), Ip6FindNeighborEntry(), Ip6FreeLinkTxToken(), Ip6GetMulticastMac(), IP6_LINK_TX_TOKEN::Link, _IP6_NEIGHBOR_ENTRY::LinkAddress, _IP6_SERVICE::LinkLocalOk, _IP6_SERVICE::Mnp, IP6_LINK_TX_TOKEN::MnpToken, _IP6_SERVICE::SnpMode, _IP6_NEIGHBOR_ENTRY::State, and _IP6_NEIGHBOR_ENTRY::Ticks.

Referenced by Ip6Output().

```
EFI_STATUS Ip6SetAddress ( IN IP6_INTERFACE *        Interface,
                           IN EFI_IPv6_ADDRESS *     Ip6Addr,
                           IN BOOLEAN                IsAnycast,
                           IN UINT8                  PrefixLength,
                           IN UINT32                 ValidLifetime,
                           IN UINT32                 PreferredLifetime,
                           IN IP6_DAD_CALLBACK       DadCallback,
                           IN VOID *                 Context
                         )
```

Set the interface's address. This will trigger the DAD process for the address to set. To set an already set address, the lifetimes wil be updated to the new value passed in.

Parameters:

[in]	*Interface*	The interface to set the address.
[in]	*Ip6Addr*	The interface's to be assigned IPv6 address.
[in]	*IsAnycast*	If TRUE, the unicast IPv6 address is anycast. Otherwise, it is not anycast.
[in]	*PrefixLength*	The prefix length of the Ip6Addr.
[in]	*ValidLifetime*	The valid lifetime for this address.
[in]	*PreferredLifetime*	The preferred lifetime for this address.
[in]	*DadCallback*	The caller's callback to trigger when DAD finishes. This is an optional parameter that may be NULL.
[in]	*Context*	The context that will be passed to DadCallback. This is an optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The interface is scheduled to be configured with the specified address.
EFI_OUT_OF_RESOURCES	Failed to set the interface's address due to lack of resources.

References _IP6_ADDRESS_INFO::Address, _IP6_DELAY_JOIN_LIST::AddressInfo, _IP6_SERVICE::AutonomousPrefix, _IP6_DELAY_JOIN_LIST::Context, _IP6_DELAY_JOIN_LIST::DadCallback, _IP6_DELAY_JOIN_LIST::DelayTime, _IP6_DELAY_JOIN_LIST::Interface, IP6_ADDR_INFO_SIGNATURE, IP6_INTERFACE_SIGNATURE, IP6_LINK_LOCAL_PREFIX_LENGTH, IP6_ONE_SECOND_IN_MS, IP6_TIMER_INTERVAL_IN_MS, Ip6IsOneOfSetAddress(), _IP6_ADDRESS_INFO::IsAnycast, _IP6_DELAY_JOIN_LIST::Link, _IP6_SERVICE::OnlinkPrefix, _IP6_ADDRESS_INFO::PreferredLifetime, _IP6_PREFIX_LIST_ENTRY::Prefix, _IP6_PREFIX_LIST_ENTRY::PrefixLength, _IP6_ADDRESS_INFO::PrefixLength, _IP6_ADDRESS_INFO::Signature, and _IP6_ADDRESS_INFO::ValidLifetime.

Referenced by Ip6ConfigOnDhcp6Event(), Ip6ConfigSetMaunualAddress(), Ip6CreateInterface(), and Ip6ProcessRouterAdvertise().

VOID EFIAPI Ip6TimerTicking (	IN EFI_EVENT	*Event*,	
	IN VOID *	*Context*	
)			

The heartbeat timer of IP6 service instance. It times out all of its IP6 children's received-but-not-delivered and transmitted-but-not-recycle packets.

Parameters:

[in]	*Event*	The IP6 service instance's heartbeat timer.
[in]	*Context*	The IP6 service instance.

References IP6_SERVICE_SIGNATURE, Ip6MldTimerTicking(), Ip6NdTimerTicking(), and Ip6PacketTimerTicking().

Referenced by Ip6CreateService().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Ip6Dxe/Ip6If.h File Reference

Data Structures

struct IP6_LINK_RX_
struct IP6_LINK_TX_
struct _IP6_ADDRES
struct _IP6_INTERFA

Defines

"memItemRight" valign="bottom">IP6_LINK_RX_SIGNATURE SIGNATURE_32 ('I', 'P', '6', 'R')
#define IP6_LINK_TX_
('I', 'P', '6', 'T')
#define IP6_INTERFAC
('I', 'P', '6', 'I')
#define IP6_ADDR_IN
('I', 'P', 'A', 'I')

Typedefs

typedef VOID(* IP6_FRAME_C
)(NET_BUF *P
EFI_STATUS I
LinkFlag, VOID
typedef BOOLEAN(* IP6_FRAME_T
)(IP6_LINK_TX
*Frame, VOID

Functions

IP6_INTERFACE * Ip6CreateInterfa
IP6_SERVICE
BOOLEAN Lin
VOID Ip6CleanInterfa
IP6_INTERFAC
IP6_PROTOCO
VOID Ip6FreeLinkTx
IP6_LINK_TX_
VOID EFIAPI Ip6OnFrameRe
EFI_EVENT E
*Context)
EFI_STATUS Ip6ReceiveFram
IP6_FRAME_C
CallBack, IN IP
*IpSb)
EFI_STATUS Ip6SendFrame (
IP6_INTERFAC
IP6_PROTOCO
NET_BUF *Pa

EFI_IPv6_ADDRESS *
IN IP6_FRAME_CALL
CallBack, IN VOID *C
VOID EFIAPI Ip6TimerTicking (IN
EFI_EVENT Event, IN
*Context)

Detailed Description

Definition for IP6 pesudo interface structure.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Ip6BuildEfiAddressList(), Ip6CandidateSource(), Ip6ConfigSetMaunualAddress(), Ip6IsOneOfSetAddress(), Ip6RemoveAddr(), Ip6SelectSourceAddress(), and Ip6SetAddress().

#define IP6_INTERFACE_SIGNATURE SIGNATURE_32 ('I', 'P', '6', 'I')

Referenced by Ip6CleanInterface(), Ip6ConfigOnPolicyChanged(), Ip6ConfigSetMaunualAddress(), Ip6CreateInterface(), Ip6DestroyPrefixListEntry(), Ip6InitDADProcess(), Ip6IsOneOfSetAddress(), and Ip6SetAddress().

#define IP6_LINK_RX_SIGNATURE SIGNATURE_32 ('I', 'P', '6', 'R')

Referenced by Ip6CreateService(), and Ip6OnFrameReceivedDpc().

#define IP6_LINK_TX_SIGNATURE SIGNATURE_32 ('I', 'P', '6', 'T')

Referenced by Ip6CreateLinkTxToken(), Ip6FreeLinkTxToken(), and Ip6OnFrameSentDpc().

Typedef Documentation

typedef VOID(* IP6_FRAME_CALLBACK)(NET_BUF *Packet, EFI_STATUS IoStatus, UINT32 LinkFlag, VOID *Context)

typedef BOOLEAN(* IP6_FRAME_TO_CANCEL)(IP6_LINK_TX_TOKEN *Frame, VOID *Context)

Function Documentation

VOID Ip6CleanInterface (IN IP6_INTERFACE * *Interface*,
IN IP6_PROTOCOL * *IpInstance*
)

Free the interface used by IpInstance. All the IP instance with the same Ip/prefix pair share the same interface. It is reference counted. All the frames that haven't been sent will be cancelled. Because the IpInstance is optional, the caller must remove IpInstance from the interface's instance list.

Parameters:

[in]	*Interface*	The interface used by the IpInstance.
[in]	*IpInstance*	The IP instance that free the interface. NULL if the IP driver is releasing the default interface.

References IP6_INTERFACE_SIGNATURE, Ip6CancelFrames(), Ip6CancelInstanceFrame(), and Ip6RemoveAddr().

Referenced by Ip6CleanProtocol(), and Ip6CleanService().

IP6_INTERFACE* Ip6CreateInterface (IN IP6_SERVICE * *IpSb*,
IN BOOLEAN *LinkLocal*
)

Create an IP6_INTERFACE.

Parameters:

[in]	*IpSb*	The IP6 service binding instance.
[in]	*LinkLocal*	If TRUE, the instance is created for link-local address. Otherwise, it is not for a link-local address.

Returns:

Point to the created IP6_INTERFACE, otherwise NULL.

References _IP6_INTERFACE::AddressCount, _IP6_INTERFACE::AddressList, _IP6_INTERFACE::ArpQues, _IP6_INTERFACE::Configured, _IP6_INTERFACE::Controller, _IP6_INTERFACE::DelayJoinList, _IP6_INTERFACE::DupAddrDetect, _IP6_INTERFACE::DupAddrDetectList, _IP6_INTERFACE::Image, IP6_INFINIT_LIFETIME, IP6_INTERFACE_SIGNATURE, IP6_LINK_LOCAL_PREFIX_LENGTH, IP6_SERVICE_SIGNATURE, Ip6CreateLinkLocalAddr(), Ip6SetAddress(), _IP6_INTERFACE::IpInstances, _IP6_INTERFACE::PromiscRecv, _IP6_INTERFACE::RefCnt, _IP6_INTERFACE::SentFrames, _IP6_INTERFACE::Service, and _IP6_INTERFACE::Signature.

Referenced by Ip6CreateService().

VOID Ip6FreeLinkTxToken (IN IP6_LINK_TX_TOKEN * *Token*)

Free the link layer transmit token. It will close the event then free the memory used.

Parameters:

[in] *Token* Token to free.

Free the link layer transmit token. It will close the event, then free the memory used.

Parameters:

[in] *Token* Token to free.

References IP6_LINK_TX_SIGNATURE.

Referenced by Ip6FreeNeighborEntry(), Ip6OnArpResolved(), Ip6OnFrameSentDpc(), and Ip6SendFrame().

```
VOID EFIAPI Ip6OnFrameReceived ( IN EFI_EVENT  Event,
                                 IN VOID *     Context
                               )
```

Request Ip6OnFrameReceivedDpc as a DPC at TPL_CALLBACK

Parameters:

Event The receive event delivered to MNP for receive.
Context Context for the callback.

Request Ip6OnFrameReceivedDpc as a DPC at TPL_CALLBACK.

Parameters:

Event The receive event delivered to MNP for receive.
Context Context for the callback.

References Ip6OnFrameReceivedDpc().

Referenced by Ip6CreateService().

```
EFI_STATUS Ip6ReceiveFrame ( IN IP6_FRAME_CALLBACK  CallBack,
                             IN IP6_SERVICE *       IpSb
                           )
```

Request to receive the packet from the interface.

Parameters:

[in] *CallBack* Function to call when the receive finished.
[in] *IpSb* Points to the IP6 service binding instance.

Return values:

EFI_ALREADY_STARTED There is already a pending receive request.
EFI_OUT_OF_RESOURCES Failed to allocate resources to receive.
EFI_SUCCESS The recieve request has been started.

Request to receive the packet from the interface.

Parameters:

[in] *CallBack* Function to call when receive finished.
[in] *IpSb* Points to IP6 service binding instance.

Return values:

EFI_ALREADY_STARTED	There is already a pending receive request.
EFI_OUT_OF_RESOURCES	Failed to allocate resource to receive.
EFI_SUCCESS	The recieve request has been started.

References IP6_LINK_RX_TOKEN::CallBack, IP6_LINK_RX_TOKEN::Context, IP6_SERVICE_SIGNATURE, and IP6_LINK_RX_TOKEN::MnpToken.

Referenced by Ip6AcceptFrame(), and Ip6DriverBindingStart().

```
EFI_STATUS Ip6SendFrame ( IN IP6_INTERFACE *          Interface,
                          IN IP6_PROTOCOL *           IpInstance,
                          IN NET_BUF *                Packet,
                          IN EFI_IPv6_ADDRESS *       NextHop,
                          IN IP6_FRAME_CALLBACK       CallBack,
                          IN VOID *                   Context
                        )
```

Send a frame from the interface. If the next hop is multicast address, it is transmitted immediately. If the next hop is a unicast, and the NextHop's MAC is not known, it will perform address resolution. If some error happened, the CallBack won't be called. So, the caller must test the return value, and take action when there is an error.

Parameters:

[in]	*Interface*	The interface to send the frame from
[in]	*IpInstance*	The IP child that request the transmission. NULL if it is the IP6 driver itself.
[in]	*Packet*	The packet to transmit.
[in]	*NextHop*	The immediate destination to transmit the packet to.
[in]	*CallBack*	Function to call back when transmit finished.
[in]	*Context*	Opaque parameter to the call back.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate resource to send the frame.
EFI_NO_MAPPING	Can't resolve the MAC for the nexthop.
EFI_SUCCESS	The packet successfully transmitted.

Send a frame from the interface. If the next hop is a multicast address, it is transmitted immediately. If the next hop is a unicast, and the NextHop's MAC is not known, it will perform address resolution. If an error occurred, the CallBack won't be called. So, the caller must test the return value, and take action when there is an error.

Parameters:

[in]	*Interface*	The interface to send the frame from
[in]	*IpInstance*	The IP child that request the transmission. NULL if it is the IP6 driver itself.
[in]	*Packet*	The packet to transmit.
[in]	*NextHop*	The immediate destination to transmit the packet to.
[in]	*CallBack*	Function to call back when transmit finished.
[in]	*Context*	Opaque parameter to the callback.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate resource to send the frame.
EFI_NO_MAPPING	Can't resolve the MAC for the nexthop.

EFI_SUCCESS	The packet successfully transmitted.

References _IP6_NEIGHBOR_ENTRY::ArpFree, _IP6_NEIGHBOR_ENTRY::ArpList, IP6_LINK_TX_TOKEN::DstMac, _IP6_NEIGHBOR_ENTRY::Frames, _IP6_NEIGHBOR_ENTRY::Interface, IP6_DELAY_FIRST_PROBE_TIME, IP6_GET_TICKS, IP6_SERVICE_SIGNATURE, Ip6CreateLinkTxToken(), Ip6FindNeighborEntry(), Ip6FreeLinkTxToken(), Ip6GetMulticastMac(), IP6_LINK_TX_TOKEN::Link, _IP6_NEIGHBOR_ENTRY::LinkAddress, _IP6_SERVICE::LinkLocalOk, _IP6_SERVICE::Mnp, IP6_LINK_TX_TOKEN::MnpToken, _IP6_SERVICE::SnpMode, _IP6_NEIGHBOR_ENTRY::State, and _IP6_NEIGHBOR_ENTRY::Ticks.

Referenced by Ip6Output().

```
VOID EFIAPI Ip6TimerTicking (  IN EFI_EVENT   Event,
                               IN VOID *      Context
                            )
```

The heartbeat timer of IP6 service instance. It times out all of its IP6 children's received-but-not-delivered and transmitted-but-not-recycle packets.

Parameters:

[in]	*Event*	The IP6 service instance's heart beat timer.
[in]	*Context*	The IP6 service instance.

The heartbeat timer of IP6 service instance. It times out all of its IP6 children's received-but-not-delivered and transmitted-but-not-recycle packets.

Parameters:

[in]	*Event*	The IP6 service instance's heartbeat timer.
[in]	*Context*	The IP6 service instance.

References IP6_SERVICE_SIGNATURE, Ip6MldTimerTicking(), Ip6NdTimerTicking(), and Ip6PacketTimerTicking().

Referenced by Ip6CreateService().

NetworkPkg/Ip6Dxe/Ip6Impl.c File Reference

Functions

EFI_STATUS EFIAPI EfiIp6GetModeData (IN EFI_IP6_PROTOCOL *This, OUT EFI_IP6_MODE_DATA *Ip6ModeData, OUT EFI_MANAGED_NETWORK_CONFIG_DATA *MnpConfigData, OUT EFI_SIMPLE_NETWORK_MODE *SnpModeData)

BOOLEAN Ip6IsValidAddress (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *Ip, IN BOOLEAN Flag)

BOOLEAN Ip6IsIllegalProtocol (IN UINT8 Protocol)

VOID Ip6InitProtocol (IN IP6_SERVICE *IpSb, IN OUT IP6_PROTOCOL *IpInstance)

EFI_STATUS Ip6ConfigProtocol (IN OUT IP6_PROTOCOL *IpInstance, IN EFI_IP6_CONFIG_DATA *Config)

EFI_STATUS Ip6CleanProtocol (IN OUT IP6_PROTOCOL *IpInstance)

EFI_STATUS Ip6ServiceConfigMnp (IN IP6_SERVICE *IpSb, IN BOOLEAN Force)

EFI_STATUS EFIAPI EfiIp6Configure (IN EFI_IP6_PROTOCOL *This, IN EFI_IP6_CONFIG_DATA *Ip6ConfigData)

EFI_STATUS EFIAPI EfiIp6Groups (IN EFI_IP6_PROTOCOL *This, IN BOOLEAN JoinFlag, IN EFI_IPv6_ADDRESS *GroupAddress)

EFI_STATUS EFIAPI EfiIp6Routes (IN EFI_IP6_PROTOCOL *This, IN BOOLEAN DeleteRoute, IN EFI_IPv6_ADDRESS *Destination, IN UINT8 PrefixLength, IN EFI_IPv6_ADDRESS *GatewayAddress)

EFI_STATUS EFIAPI EfiIp6Neighbors (IN EFI_IP6_PROTOCOL *This, IN BOOLEAN DeleteFlag, IN EFI_IPv6_ADDRESS *TargetIp6Address, IN EFI_MAC_ADDRESS *TargetLinkAddress, IN UINT32 Timeout, IN BOOLEAN Override)

EFI_STATUS EFIAPI Ip6TokenExist (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Context)

EFI_STATUS Ip6TxTokenValid (IN EFI_IP6_COMPLETION_TOKEN *Token)

VOID EFIAPI Ip6FreeTxToken (IN VOID *Context)

VOID Ip6OnPacketSent (IN NET_BUF *Packet, IN EFI_STATUS IoStatus, IN UINT32 Flag, IN VOID *Context)

EFI_STATUS EFIAPI EfiIp6Transmit (IN EFI_IP6_PROTOCOL *This, IN EFI_IP6_COMPLETION_TOKEN *Token)

EFI_STATUS EFIAPI EfiIp6Receive (IN EFI_IP6_PROTOCOL *This, IN EFI_IP6_COMPLETION_TOKEN *Token)

EFI_STATUS EFIAPI Ip6CancelTxTokens (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Context)

EFI_STATUS EFIAPI Ip6CancelRxTokens (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Context)

EFI_STATUS Ip6Cancel (IN IP6_PROTOCOL *IpInstance, IN EFI_IP6_COMPLETION_TOKEN *Token)

EFI_STATUS EFIAPI EfiIp6Cancel (IN EFI_IP6_PROTOCOL *This, IN EFI_IP6_COMPLETION_TOKEN *Token)

EFI_STATUS EFIAPI EfiIp6Poll (IN EFI_IP6_PROTOCOL *This)

Variables

EFI_IPSEC2_PROTOCOL * mIpSec = NULL

EFI_IP6_PROTOCOL mEfiIp6ProtocolTemplete

Detailed Description

Implementation of EFI_IP6_PROTOCOL protocol interfaces.

(C) Copyright 2014 Hewlett-Packard Development Company, L.P.
Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI EfiIp6Cancel ( IN EFI_IP6_PROTOCOL *              This,
                                 IN EFI_IP6_COMPLETION_TOKEN *      Token
                               )
```

Abort an asynchronous transmit or receive request.

The Cancel() function is used to abort a pending transmit or receive request. If the token is in the transmit or receive request queues, after calling this function, Token->Status will be set to EFI_ABORTED, and then Token->Event will be signaled. If the token is not in one of the queues, which usually means the asynchronous operation has completed, this function will not signal the token, and EFI_NOT_FOUND is returned.

Parameters:

[in]	*This*	Pointer to the EFI_IP6_PROTOCOL instance.
[in]	*Token*	Pointer to a token that has been issued by EFI_IP6_PROTOCOL.Transmit() or EFI_IP6_PROTOCOL.Receive(). If NULL, all pending tokens are aborted. Type EFI_IP6_COMPLETION_TOKEN is defined in EFI_IP6_PROTOCOL.Transmit().

Return values:

EFI_SUCCESS	The asynchronous I/O request was aborted and Token->Event was signaled. When Token is NULL, all pending requests were aborted, and their events were signaled.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This instance has not been started.
EFI_NOT_FOUND	When Token is not NULL, the asynchronous I/O request was not found in the transmit or receive queue. It has either completed or was not issued by Transmit() and Receive().
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References IP6_INSTANCE_FROM_PROTOCOL, IP6_STATE_CONFIGED, Ip6Cancel(), and _IP6_PROTOCOL::State.

```
EFI_STATUS EFIAPI EfiIp6Configure ( IN EFI_IP6_PROTOCOL *          This,
```

IN EFI_IP6_CONFIG_DATA * *Ip6ConfigData*
)

Assigns an IPv6 address and subnet mask to this EFI IPv6 Protocol driver instance.

The Configure() function is used to set, change, or reset the operational parameters and filter settings for this EFI IPv6 Protocol instance. Until these parameters have been set, no network traffic can be sent or received by this instance. Once the parameters have been reset (by calling this function with Ip6ConfigData set to NULL), no more traffic can be sent or received until these parameters have been set again. Each EFI IPv6 Protocol instance can be started and stopped independently of each other by enabling or disabling their receive filter settings with the Configure() function.

If Ip6ConfigData.StationAddress is a valid non-zero IPv6 unicast address, it is required to be one of the currently configured IPv6 addresses listed in the EFI IPv6 drivers, or else EFI_INVALID_PARAMETER will be returned. If Ip6ConfigData.StationAddress is unspecified, the IPv6 driver will bind a source address according to the source address selection algorithm. Clients could frequently call GetModeData() to check get currently configured IPv6 address list in the EFI IPv6 driver. If both Ip6ConfigData.StationAddress and Ip6ConfigData.Destination are unspecified, when transmitting the packet afterwards, the source address filled in each outgoing IPv6 packet is decided based on the destination of this packet.

If operational parameters are reset or changed, any pending transmit and receive requests will be cancelled. Their completion token status will be set to EFI_ABORTED and their events will be signaled.

Parameters:

`[in]`	*This*	Pointer to the EFI_IP6_PROTOCOL instance.
`[in]`	*Ip6ConfigData*	Pointer to the EFI IPv6 Protocol configuration data structure. If NULL, reset the configuration data.

Return values:

EFI_SUCCESS	The driver instance was successfully opened.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Ip6ConfigData.StationAddress is neither zero nor a unicast IPv6 address. ◊ Ip6ConfigData.StationAddress is neither zero nor one of the configured IP addresses in the EFI IPv6 driver. ◊ Ip6ConfigData.DefaultProtocol is illegal.
EFI_OUT_OF_RESOURCES	The EFI IPv6 Protocol driver instance data could not be allocated.
EFI_NO_MAPPING	The IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_ALREADY_STARTED	The interface is already open and must be stopped before the IPv6 address or prefix length can be changed.
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI IPv6 Protocol driver instance was not opened.
EFI_UNSUPPORTED	Default protocol specified through Ip6ConfigData.DefaulProtocol isn't supported.

References _IP6_PROTOCOL::ConfigData, IP6_INSTANCE_FROM_PROTOCOL, IP6_NO_MAPPING, IP6_STATE_CONFIGED, IP6_STATE_UNCONFIGED, Ip6CleanProtocol(), Ip6ConfigProtocol(), Ip6IsIllegalProtocol(), Ip6IsValidAddress(), Ip6ServiceConfigMnp(), _IP6_SERVICE::LinkLocalDadFail, _IP6_PROTOCOL::Service, and _IP6_PROTOCOL::State.

(IN EFI_IP6_PROTOCOL * *This*,

```
EFI_STATUS EFIAPI
EfiIp6GetModeData
                                OUT EFI_IP6_MODE_DATA *                    Ip6ModeData,
                                OUT
                                EFI_MANAGED_NETWORK_CONFIG_DATA *          MnpConfigData,
                                OUT EFI_SIMPLE_NETWORK_MODE *              SnpModeData
                              )
```

Gets the current operational settings for this instance of the EFI IPv6 Protocol driver.

The GetModeData() function returns the current operational mode data for this driver instance. The data fields in EFI_IP6_MODE_DATA are read only. This function is used optionally to retrieve the operational mode data of underlying networks or drivers.

Parameters:

[in]	*This*	Pointer to the EFI_IP6_PROTOCOL instance.
[out]	*Ip6ModeData*	Pointer to the EFI IPv6 Protocol mode data structure.
[out]	*MnpConfigData*	Pointer to the managed network configuration data structure.
[out]	*SnpModeData*	Pointer to the simple network mode data structure.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	This is NULL.
EFI_OUT_OF_RESOURCES	The required mode data could not be allocated.

References _IP6_PROTOCOL::ConfigData, _IP6_INTERFACE::Configured, _IP6_PROTOCOL::GroupCount, _IP6_PROTOCOL::GroupList, _IP6_PROTOCOL::Interface, IP6_INSTANCE_FROM_PROTOCOL, IP6_STATE_CONFIGED, Ip6BuildEfiAddressList(), Ip6BuildEfiNeighborCache(), Ip6BuildEfiRouteTable(), Ip6BuildPrefixTable(), _IP6_SERVICE::LinkLocalDadFail, _IP6_SERVICE::MaxPacketSize, mIp6SupportedIcmp, _IP6_SERVICE::Mnp, _IP6_SERVICE::RouteTable, _IP6_PROTOCOL::Service, and _IP6_PROTOCOL::State.

```
EFI_STATUS EFIAPI EfiIp6Groups  ( IN EFI_IP6_PROTOCOL *   This,
                                  IN BOOLEAN              JoinFlag,
                                  IN EFI_IPv6_ADDRESS *   GroupAddress
                                )
```

Joins and leaves multicast groups.

The Groups() function is used to join and leave multicast group sessions. Joining a group will enable reception of matching multicast packets. Leaving a group will disable reception of matching multicast packets. Source-Specific Multicast isn't required to be supported.

If JoinFlag is FALSE and GroupAddress is NULL, all joined groups will be left.

Parameters:

[in]	*This*	Pointer to the EFI_IP6_PROTOCOL instance.
[in]	*JoinFlag*	Set to TRUE to join the multicast group session, and FALSE to leave.
[in]	*GroupAddress*	Pointer to the IPv6 multicast address. This is an optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	One or more of the following is TRUE: ◊ This is NULL. ◊ JoinFlag is TRUE and GroupAddress is NULL. ◊ GroupAddress is not NULL and *GroupAddress is not a multicast IPv6 address. ◊ GroupAddress is not NULL and *GroupAddress is in the range of SSM destination address.
EFI_NOT_STARTED	This instance has not been started.
EFI_OUT_OF_RESOURCES	System resources could not be allocated.
EFI_UNSUPPORTED	This EFI IPv6 Protocol implementation does not support multicast groups.
EFI_ALREADY_STARTED	The group address is already in the group table (when JoinFlag is TRUE).
EFI_NOT_FOUND	The group address is not in the group table (when JoinFlag is FALSE).
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References IP6_INSTANCE_FROM_PROTOCOL, IP6_STATE_CONFIGED, Ip6Groups(), _IP6_SERVICE::LinkLocalDadFail, _IP6_PROTOCOL::Service, and _IP6_PROTOCOL::State.

```
EFI_STATUS EFIAPI EfiIp6Neighbors ( IN EFI_IP6_PROTOCOL *   This,
                                    IN BOOLEAN              DeleteFlag,
                                    IN EFI_IPv6_ADDRESS *   TargetIp6Address,
                                    IN EFI_MAC_ADDRESS *    TargetLinkAddress,
                                    IN UINT32               Timeout,
                                    IN BOOLEAN              Override
                                  )
```

Add or delete Neighbor cache entries.

The Neighbors() function is used to add, update, or delete an entry from neighbor cache. IPv6 neighbor cache entries are typically inserted and updated by the network protocol driver as network traffic is processed. Most neighbor cache entries will timeout and be deleted if the network traffic stops. Neighbor cache entries that were inserted by Neighbors() may be static (will not timeout) or dynamic (will timeout).

The implementation should follow the neighbor cache timeout mechanism which is defined in RFC4861. The default neighbor cache timeout value should be tuned for the expected network environment

Parameters:

[in] *This*	Pointer to the EFI_IP6_PROTOCOL instance.
[in] *DeleteFlag*	Set to TRUE to delete the specified cache entry, set to FALSE to add (or update, if it already exists and Override is TRUE) the specified cache entry. TargetIp6Address is used as the key to find the requested cache entry.
[in] *TargetIp6Address*	Pointer to the Target IPv6 address.
[in] *TargetLinkAddress*	Pointer to the link-layer address of the target. Ignored if NULL.
[in] *Timeout*	Time in 100-ns units that this entry will remain in the neighbor cache, it will be deleted after Timeout. A value of zero means that the entry is permanent. A non-zero value means that the entry is dynamic.
[in] *Override*	If TRUE, the cached link-layer address of the matching entry will be overridden and updated; if FALSE, EFI_ACCESS_DENIED will be

returned if a corresponding cache entry already existed.

Return values:

EFI_SUCCESS	The data has been queued for transmission.
EFI_NOT_STARTED	This instance has not been started.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ TargetIpAddress is NULL. ◊ *TargetLinkAddress is invalid when not NULL. ◊ *TargetIpAddress is not a valid unicast IPv6 address. ◊ *TargetIpAddress is one of the local configured IPv6 addresses.
EFI_OUT_OF_RESOURCES	Could not add the entry to the neighbor cache.
EFI_NOT_FOUND	This entry is not in the neighbor cache (when DeleteFlag is TRUE or when DeleteFlag is FALSE while TargetLinkAddress is NULL.).
EFI_ACCESS_DENIED	The to-be-added entry is already defined in the neighbor cache, and that entry is tagged as un-overridden (when Override is FALSE).

References IP6_INSTANCE_FROM_PROTOCOL, IP6_STATE_CONFIGED, Ip6AddNeighbor(), Ip6DelNeighbor(), Ip6IsOneOfSetAddress(), Ip6IsValidAddress(), Ip6IsValidLinkAddress(), _IP6_SERVICE::LinkLocalDadFail, _IP6_PROTOCOL::Service, and _IP6_PROTOCOL::State.

EFI_STATUS EFIAPI EfiIp6Poll (IN EFI_IP6_PROTOCOL * *This*)

Polls for incoming data packets, and processes outgoing data packets.

The Poll() function polls for incoming data packets and processes outgoing data packets. Network drivers and applications can call the EFI_IP6_PROTOCOL.Poll() function to increase the rate that data packets are moved between the communications device and the transmit and receive queues.

In some systems the periodic timer event may not poll the underlying communications device fast enough to transmit and/or receive all data packets without missing incoming packets or dropping outgoing packets. Drivers and applications that are experiencing packet loss should try calling the EFI_IP6_PROTOCOL.Poll() function more often.

Parameters:

`[in]` *This* Pointer to the EFI_IP6_PROTOCOL instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_NOT_STARTED	This EFI IPv6 Protocol instance has not been started.
EFI_INVALID_PARAMETER	This is NULL.
EFI_DEVICE_ERROR	An unexpected system error or network error occurred.
EFI_NOT_READY	No incoming or outgoing data was processed.
EFI_TIMEOUT	Data was dropped out of the transmit and/or receive queue. Consider increasing the polling rate.

References IP6_INSTANCE_FROM_PROTOCOL, IP6_STATE_UNCONFIGED, _IP6_SERVICE::LinkLocalDadFail, _IP6_SERVICE::Mnp, _IP6_PROTOCOL::Service, and _IP6_PROTOCOL::State.

EFI_STATUS EFIAPI EfiIp6Receive (IN EFI_IP6_PROTOCOL * *This*,

```
                                    IN EFI_IP6_COMPLETION_TOKEN *   Token
                                  )
```

Places a receiving request into the receiving queue.

The Receive() function places a completion token into the receive packet queue. This function is always asynchronous.

The Token.Event field in the completion token must be filled in by the caller and cannot be NULL. When the receive operation completes, the EFI IPv6 Protocol driver updates the Token.Status and Token.Packet.RxData fields and the Token.Event is signaled.

Current Udp implementation creates an IP child for each Udp child. It initates a asynchronous receive immediately no matter whether there is no mapping or not. Therefore, disable the returning EFI_NO_MAPPING for now. To enable it, the following check must be performed:

if (NetIp6IsUnspecifiedAddr (&Config->StationAddress) && IP6_NO_MAPPING (IpInstance)) { Status = EFI_NO_MAPPING; goto Exit; }

Parameters:

`[in]` *This* Pointer to the EFI_IP6_PROTOCOL instance.

`[in]` *Token* Pointer to a token that is associated with the receive data descriptor.

Return values:

EFI_SUCCESS	The receive completion token was cached.
EFI_NOT_STARTED	This EFI IPv6 Protocol instance has not been started.
EFI_NO_MAPPING	When IP6 driver responsible for binding source address to this instance, while no source address is available for use.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token.Event is NULL.
EFI_OUT_OF_RESOURCES	The receive completion token could not be queued due to a lack of system resources (usually memory).
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI IPv6 Protocol instance has been reset to startup defaults.
EFI_ACCESS_DENIED	The receive completion token with the same Token.Event was already in the receive queue.
EFI_NOT_READY	The receive request could not be queued because the receive queue is full.

References IP6_INSTANCE_FROM_PROTOCOL, IP6_STATE_CONFIGED, Ip6InstanceDeliverPacket(), Ip6TokenExist(), _IP6_SERVICE::LinkLocalDadFail, _IP6_PROTOCOL::RxTokens, _IP6_PROTOCOL::Service, and _IP6_PROTOCOL::State.

```
EFI_STATUS EFIAPI EfiIp6Routes  ( IN EFI_IP6_PROTOCOL *   This,
                                  IN BOOLEAN              DeleteRoute,
                                  IN EFI_IPv6_ADDRESS *   Destination,
                                  IN UINT8                PrefixLength,
                                  IN EFI_IPv6_ADDRESS *   GatewayAddress
                                )
```

Adds and deletes routing table entries.

The Routes() function adds a route to, or deletes a route from, the routing table.

Routes are determined by comparing the leftmost PrefixLength bits of Destination with the destination IPv6 address arithmetically. The gateway address must be on the same subnet as the configured station address.

The default route is added with Destination and PrefixLegth both set to all zeros. The default route matches all destination IPv6 addresses that do not match any other routes.

All EFI IPv6 Protocol instances share a routing table.

Parameters:

[in]	*This*	Pointer to the EFI_IP6_PROTOCOL instance.
[in]	*DeleteRoute*	Set to TRUE to delete this route from the routing table. Set to FALSE to add this route to the routing table. Destination, PrefixLength and Gateway are used as the key to each route entry.
[in]	*Destination*	The address prefix of the subnet that needs to be routed. This is an optional parameter that may be NULL.
[in]	*PrefixLength*	The prefix length of Destination. Ignored if Destination is NULL.
[in]	*GatewayAddress*	The unicast gateway IPv6 address for this route. This is an optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_STARTED	The driver instance has not been started.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ When DeleteRoute is TRUE, both Destination and GatewayAddress are NULL. ◊ When DeleteRoute is FALSE, either Destination or GatewayAddress is NULL. ◊ *GatewayAddress is not a valid unicast IPv6 address. ◊ *GatewayAddress is one of the local configured IPv6 addresses.
EFI_OUT_OF_RESOURCES	Could not add the entry to the routing table.
EFI_NOT_FOUND	This route is not in the routing table (when DeleteRoute is TRUE).
EFI_ACCESS_DENIED	The route is already defined in the routing table (when DeleteRoute is FALSE).

References _IP6_PROTOCOL::ConfigData, IP6_INSTANCE_FROM_PROTOCOL, IP6_STATE_CONFIGED, Ip6AddRoute(), Ip6DelRoute(), Ip6IsValidAddress(), _IP6_SERVICE::LinkLocalDadFail, _IP6_SERVICE::RouteTable, _IP6_PROTOCOL::Service, and _IP6_PROTOCOL::State.

```
EFI_STATUS EFIAPI EfiIp6Transmit ( IN EFI_IP6_PROTOCOL *              This,
                                   IN EFI_IP6_COMPLETION_TOKEN *      Token
                                 )
```

Places outgoing data packets into the transmit queue.

The Transmit() function places a sending request in the transmit queue of this EFI IPv6 Protocol instance. Whenever the packet in the token is sent out or some errors occur, the event in the token will be signaled, and the status is updated.

Parameters:

[in] *This* Pointer to the EFI_IP6_PROTOCOL instance.

[in] *Token* Pointer to the transmit token.

Return values:

EFI_SUCCESS	The data has been queued for transmission.
EFI_NOT_STARTED	This instance has not been started.
EFI_NO_MAPPING	The IPv6 driver was responsible for choosing a source address for this transmission, but no source address was available for use.
EFI_INVALID_PARAMETER	One or more of the following is TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token.Event is NULL. ◊ Token.Packet.TxData is NULL. ◊ Token.Packet.ExtHdrsLength is not zero and Token.Packet.ExtHdrs is NULL. ◊ Token.Packet.FragmentCount is zero. ◊ One or more of the Token.Packet.TxData. FragmentTable[].FragmentLength fields is zero. ◊ One or more of the Token.Packet.TxData. FragmentTable[].FragmentBuffer fields is NULL. ◊ Token.Packet.TxData.DataLength is zero or not equal to the sum of fragment lengths. ◊ Token.Packet.TxData.DestinationAddress is non zero when DestinationAddress is configured as non-zero when doing Configure() for this EFI IPv6 protocol instance. ◊ Token.Packet.TxData.DestinationAddress is unspecified when DestinationAddress is unspecified when doing Configure() for this EFI IPv6 protocol instance.
EFI_ACCESS_DENIED	The transmit completion token with the same Token. Event was already in the transmit queue.
EFI_NOT_READY	The completion token could not be queued because the transmit queue is full.
EFI_NOT_FOUND	Not route is found to destination address.
EFI_OUT_OF_RESOURCES	Could not queue the transmit data.
EFI_BUFFER_TOO_SMALL	Token.Packet.TxData.TotalDataLength is too short to transmit.
EFI_BAD_BUFFER_SIZE	If Token.Packet.TxData.DataLength is beyond the maximum that which can be described through the Fragment Offset field in Fragment header when performing fragmentation.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References _IP6_PROTOCOL::ConfigData, IP6_INSTANCE_FROM_PROTOCOL, IP6_MAX_HEADLEN, IP6_STATE_CONFIGED, IP6_US_TO_SEC, Ip6FreeTxToken(), Ip6OnPacketSent(), Ip6Output(), Ip6SelectSourceAddress(), Ip6TokenExist(), Ip6TxTokenValid(), IP6_TXTOKEN_WRAP::IpInstance, IP6_TXTOKEN_WRAP::Life, _IP6_SERVICE::LinkLocalDadFail, IP6_TXTOKEN_WRAP::Packet, IP6_TXTOKEN_WRAP::Sent, _IP6_PROTOCOL::Service, _IP6_PROTOCOL::State, IP6_TXTOKEN_WRAP::Token, and _IP6_PROTOCOL::TxTokens.

```
EFI_STATUS Ip6Cancel ( IN IP6_PROTOCOL *                IpInstance,
                       IN EFI_IP6_COMPLETION_TOKEN *    Token
                     )
```

Cancel the user's receive/transmit request. It is the worker function of EfiIp6Cancel API.

Parameters:

[in]	*IpInstance*	The IP6 child.
[in]	*Token*	The token to cancel. If NULL, all token will be cancelled.

Return values:

EFI_SUCCESS	The token is cancelled.
EFI_NOT_FOUND	The token isn't found on either the transmit/receive queue.
EFI_DEVICE_ERROR	Not all tokens are cancelled when Token is NULL.

References Ip6CancelRxTokens(), and Ip6CancelTxTokens().

Referenced by EfiIp6Cancel(), Ip6CleanProtocol(), and Ip6ConfigProtocol().

```
EFI_STATUS EFIAPI Ip6CancelRxTokens ( IN NET_MAP *        Map,
                                      IN NET_MAP_ITEM *   Item,
                                      IN VOID *           Context
                                    )
```

Cancel the receive request. This is simple, because it is only enqueued in our local receive map.

Parameters:

[in]	*Map*	The IP6 child's receive queue.
[in]	*Item*	Current receive request to cancel.
[in]	*Context*	The user's token to cancel.

Return values:

EFI_SUCCESS	Continue to check the next receive request on the queue.
EFI_ABORTED	The user's token (token != NULL) has been cancelled.

Referenced by Ip6Cancel().

```
EFI_STATUS EFIAPI Ip6CancelTxTokens ( IN NET_MAP *        Map,
                                      IN NET_MAP_ITEM *   Item,
                                      IN VOID *           Context
                                    )
```

Cancel the transmitted but not recycled packet. If a matching token is found, it will call Ip6CancelPacket to cancel the packet. Ip6CancelPacket cancels all the fragments of the packet. When all the fragments are freed, the IP6_TXTOKEN_WRAP is deleted from the Map, and user's event is signalled. Because Ip6CancelPacket and other functions are all called in line, after Ip6CancelPacket returns, the Item has been freed.

Parameters:

[in]	*Map*	The IP6 child's transmit queue.
[in]	*Item*	The current transmitted packet to test.
[in]	*Context*	The user's token to cancel.

Return values:

EFI_SUCCESS	Continue to check the next Item.

EFI_ABORTED The user's Token (Token != NULL) is cancelled.

References _IP6_PROTOCOL::Interface, Ip6CancelPacket(), IP6_TXTOKEN_WRAP::IpInstance, and IP6_TXTOKEN_WRAP::Packet.

Referenced by Ip6Cancel().

EFI_STATUS Ip6CleanProtocol (IN OUT IP6_PROTOCOL * *IpInstance*)

Clean up the IP6 child, and release all the resources used by it.

Parameters:

`[in,out]` *IpInstance* The IP6 child to clean up.

Return values:

EFI_SUCCESS The IP6 child is cleaned up.
EFI_DEVICE_ERROR Some resources failed to be released.

References Ip6Cancel(), Ip6CleanInterface(), and Ip6Groups().

Referenced by EfiIp6Configure(), Ip6ServiceBindingCreateChild(), and Ip6ServiceBindingDestroyChild().

EFI_STATUS Ip6ConfigProtocol (IN OUT IP6_PROTOCOL * *IpInstance*,
IN EFI_IP6_CONFIG_DATA * *Config*
)

Configure the IP6 child. If the child is already configured, change the configuration parameter. Otherwise, configure it for the first time. The caller should validate the configuration before deliver them to it. It also don't do configure NULL.

Parameters:

`[in,out]` *IpInstance* The IP6 child to configure.
`[in]` *Config* The configure data.

Return values:

EFI_SUCCESS The IP6 child is successfully configured.
EFI_DEVICE_ERROR Failed to free the pending transive or to configure underlying MNP, or other errors.
EFI_NO_MAPPING The IP6 child is configured to use the default address, but the default address hasn't been configured. The IP6 child doesn't need to be reconfigured when the default address is configured.
EFI_OUT_OF_RESOURCES No more memory space is available.
other Other error occurs.

References _IP6_SERVICE::DefaultInterface, IP6_LINK_LOCAL_PREFIX_LENGTH, IP6_STATE_CONFIGED, Ip6Cancel(), Ip6IsOneOfSetAddress(), Ip6SelectSourceAddress(), _IP6_INTERFACE::IpInstances, and _IP6_ADDRESS_INFO::PrefixLength.

Referenced by EfiIp6Configure().

VOID EFIAPI Ip6FreeTxToken (IN VOID * *Context*)

The callback function for the net buffer which wraps the user's transmit token. Although this function seems simple, there are some subtle aspects. When user requests the IP to transmit a packet by passing it a token, the token is wrapped in an IP6_TXTOKEN_WRAP and the data is wrapped in an net buffer. The net buffer's Free function is set to Ip6FreeTxToken. The Token and token wrap are added to the IP child's TxToken map. Then the buffer is passed to Ip6Output for transmission. If an error happened before that, the buffer is freed, which in turn frees the token wrap. The wrap may have been added to the TxToken map or not, and the user's event shouldn't be fired because we are still in the EfiIp6Transmit. If the buffer has been sent by Ip6Output, it should be removed from the TxToken map and user's event signaled. The token wrap and buffer are bound together. Check the comments in Ip6Output for information about IP fragmentation.

Parameters:

`[in]` *Context* The token's wrap.

References IP6_TXTOKEN_WRAP::IpInstance, IP6_TXTOKEN_WRAP::IpSecRecycleSignal, IP6_TXTOKEN_WRAP::Sent, IP6_TXTOKEN_WRAP::Token, and _IP6_PROTOCOL::TxTokens.

Referenced by EfiIp6Transmit(), and Ip6IpSecProcessPacket().

```
VOID Ip6InitProtocol ( IN IP6_SERVICE *          IpSb,
                       IN OUT IP6_PROTOCOL *     IpInstance
                     )
```

Intiialize the IP6_PROTOCOL structure to the unconfigured states.

Parameters:

`[in]` *IpSb* The IP6 service instance.

`[in,out]` *IpInstance* The IP6 child instance.

References IP6_PROTOCOL_SIGNATURE, IP6_STATE_UNCONFIGED, and mEfiIp6ProtocolTemplete.

Referenced by Ip6ServiceBindingCreateChild().

```
BOOLEAN Ip6IsIllegalProtocol ( IN UINT8  Protocol  )
```

Validate whether the value of protocol is illegal or not. Protocol is the 'Next Header' field in the last IPv6 extension header, or basic IPv6 header is there's no extension header.

Parameters:

`[in]` *Protocol* Default value of 'Next Header'

Return values:

TRUE The protocol is illegal.

FALSE The protocol is legal.

Referenced by EfiIp6Configure().

```
BOOLEAN Ip6IsValidAddress ( IN IP6_SERVICE *          IpSb,
                            IN EFI_IPv6_ADDRESS *     Ip,
                            IN BOOLEAN                Flag
                          )
```

Validate that Ipv6 address is OK to be used as station address or next hop address/ neighbor.

Parameters:

[in] *IpSb* The IP6 service instance.
[in] *Ip* The IPv6 address to validate.
[in] *Flag* If TRUE, validate if the address is OK to be used as station address. If FALSE, validate if the address is OK to be used as the next hop address/ neighbor.

Return values:

TRUE The Ip address is valid and could be used.
FALSE Invalid Ip address.

References Ip6IsOneOfSetAddress().

Referenced by EfiIp6Configure(), EfiIp6Neighbors(), and EfiIp6Routes().

```
VOID Ip6OnPacketSent ( IN NET_BUF *     Packet,
                       IN EFI_STATUS    IoStatus,
                       IN UINT32        Flag,
                       IN VOID *        Context
                     )
```

The callback function to Ip6Output to update the transmit status.

Parameters:

[in] *Packet* The user's transmit packet.
[in] *IoStatus* The result of the transmission.
[in] *Flag* Not used during transmission.
[in] *Context* The token's wrap.

References IP6_TXTOKEN_WRAP::Packet, and IP6_TXTOKEN_WRAP::Token.

Referenced by EfiIp6Transmit().

```
EFI_STATUS Ip6ServiceConfigMnp ( IN IP6_SERVICE *   IpSb,
                                 IN BOOLEAN         Force
                               )
```

Configure the MNP parameter used by IP. The IP driver uses one MNP child to transmit/receive frames. By default, it configures MNP to receive unicast/multicast/broadcast. Also, it will enable/disable the promiscuous receive according to whether there is IP child enable that or not. If Force is FALSE, it will iterate through all the IP children to check whether the promiscuous receive setting has been changed. If it hasn't been changed, it won't reconfigure the MNP. If Force is TRUE, the MNP is configured whether that is changed or not.

Parameters:

[in] *IpSb* The IP6 service instance that is to be changed.
[in] *Force* Force the configuration or not.

Return values:

EFI_SUCCESS The MNP successfully configured/reconfigured.
Others Configuration failed.

References _IP6_PROTOCOL::ConfigData, _IP6_INTERFACE::IpInstances, and _IP6_INTERFACE::PromiscRecv.

Referenced by EfiIp6Configure(), and Ip6CreateService().

EFI_STATUS EFIAPI Ip6TokenExist (IN NET_MAP * *Map*,
IN NET_MAP_ITEM * *Item*,
IN VOID * *Context*
)

Check whether the user's token or event has already been enqueue on IP6's list.

Parameters:

[in]	*Map*	The container of either user's transmit or receive token.
[in]	*Item*	Current item to check against.
[in]	*Context*	The Token to check againist.

Return values:

EFI_ACCESS_DENIED	The token or event has already been enqueued in IP
EFI_SUCCESS	The current item isn't the same token/event as the context.

Referenced by EfiIp6Receive(), and EfiIp6Transmit().

EFI_STATUS Ip6TxTokenValid (IN EFI_IP6_COMPLETION_TOKEN * *Token*)

Validate the user's token against the current station address.

Parameters:

[in]	*Token*	User's token to validate.

Return values:

EFI_INVALID_PARAMETER	Some parameters are invalid.
EFI_BAD_BUFFER_SIZE	The user's option/data is too long.
EFI_SUCCESS	The token is OK.

Referenced by EfiIp6Transmit().

Variable Documentation

EFI_IP6_PROTOCOL mEfiIp6ProtocolTemplete

Initial value:

```
{
 EfiIp6GetModeData,
 EfiIp6Configure,
 EfiIp6Groups,
 EfiIp6Routes,
 EfiIp6Neighbors,
 EfiIp6Transmit,
 EfiIp6Receive,
 EfiIp6Cancel,
 EfiIp6Poll
}
```

Referenced by Ip6InitProtocol().

EFI_IPSEC2_PROTOCOL* mIpSec = NULL

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Ip6Dxe/Ip6Impl.h File Reference

Data Structures

struct IP6_TXTOK
struct IP6_IPSEC_
struct IP6_RXDAT
struct _IP6_PROTC
struct _IP6_SERVI

Defines

"memItemRight" valign="bottom">IP6_PROTOCOL_SIGNATURE SIGNATURE_32 ('I', 'P', '6', 'P')
#define IP6_SERVIC
#define IP6_STATE_
#define IP6_STATE_
#define IP6_SERVIC
#define IP6_SERVIC
#define IP6_SERVIC
#define IP6_SERVIC
#define IP6_INSTAN
IP6_PROTOC
#define IP6_SERVIC
ServiceBindi
#define IP6_NO_MA

Functions

VOID EFIAPI Ip6FreeTxTo
EFI_STATUS Ip6ServiceCo
EFI_STATUS Ip6Cancel (IN
EFI_IP6_CO
VOID Ip6InitProtoc
*IpInstance)
EFI_STATUS Ip6CleanProt
EFI_STATUS EFIAPI EfiIp6GetMo
EFI_IP6_MO
EFI_MANAC
OUT EFI_SI
EFI_STATUS EFIAPI EfiIp6Config
EFI_IP6_CO
EFI_STATUS EFIAPI EfiIp6Groups
IN EFI_IPv6_
EFI_STATUS EFIAPI EfiIp6Routes
DeleteRoute,
PrefixLength,
EFI_STATUS EFIAPI EfiIp6Neighb
DeleteFlag, I
EFI_MAC_A
BOOLEAN C
EFI_STATUS EFIAPI

EfiIp6Transmit (IN E
EFI_IP6_COMPLET
EFI_STATUS EFIAPI EfiIp6Receive (IN EF
EFI_IP6_COMPLET
EFI_STATUS EFIAPI EfiIp6Cancel (IN EFI
EFI_IP6_COMPLET
EFI_STATUS EFIAPI EfiIp6Poll (IN EFI_I

Variables

EFI_IPSEC2_PROTOCOL * mIpSec
BOOLEAN mIpSec2Installed

Detailed Description

Implementation of EFI_IP6_PROTOCOL protocol interfaces and type definitions.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.
(C) Copyright 2015 Hewlett-Packard Development Company, L.P.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"paramtype">Ip6) CR ((Ip6), IP6_PROTOCOL, Ip6Proto, IP6_PROTOCOL_SIGNATURE)

Referenced by EfiIp6Cancel(), EfiIp6Configure(), EfiIp6GetModeData(), EfiIp6Groups(), EfiIp6Neighbors(), EfiIp6Poll(), EfiIp6Receive(), EfiIp6Routes(), EfiIp6Transmit(), and Ip6ServiceBindingDestroyChild().

#define IP6_NO_MAPPING (IpInstance) (!(IpInstance)->Interface->Configured)

Referenced by EfiIp6Configure().

#define IP6_PROTOCOL_SIGNATURE SIGNATURE_32 ('I', 'P', '6', 'P')

Referenced by Ip6BuildEfiNeighborCache(), Ip6BuildPrefixTable(), Ip6CombineGroups(), Ip6DestroyChildEntryByAddr(), Ip6DestroyChildEntryInHandleBuffer(), Ip6InitProtocol(), Ip6InterfaceEnquePacket(), and Ip6IsValidProtocol().

#define IP6_SERVICE_CONFIGED 2

#define IP6_SERVICE_DESTROY 3

Referenced by Ip6AcceptFrame(), and Ip6DriverBindingStop().

#define IP6_SERVICE_FROM_PROTOCOL (Sb) CR ((Sb), IP6_SERVICE, ServiceBinding, IP6_SERVICE_SIGNATURE)

Referenced by Ip6DriverBindingStop(), Ip6ServiceBindingCreateChild(), and Ip6ServiceBindingDestroyChild().

#define IP6_SERVICE_SIGNATURE SIGNATURE_32 ('I', 'P', '6', 'S')

Referenced by EfiIp6ConfigSetData(), Ip6AcceptFrame(), Ip6CancelFrames(), Ip6CreateDefaultRouter(), Ip6CreateInterface(), Ip6CreateInterfaceID(), Ip6CreateLinkLocalAddr(), Ip6CreateMldEntry(), Ip6CreateNeighborEntry(), Ip6CreatePrefixListEntry(), Ip6CreateService(), Ip6DestroyInstanceByAddress(), Ip6FindDefaultRouter(), Ip6FindMldEntry(), Ip6FindNeighborEntry(), Ip6FindPrefixListEntry(), Ip6IsValidProtocol(), Ip6NdFasterTimerTicking(), Ip6OnArpResolved(), Ip6OnFrameReceivedDpc(), Ip6Output(), Ip6ProcessIcmpInformation(), Ip6ReceiveFrame(), Ip6RemoveAddr(), Ip6SelectInterface(), Ip6SelectSourceAddress(), Ip6SendFrame(), Ip6SendMldDone(), Ip6SendMldReport(), Ip6SendNeighborAdvertise(), Ip6SendNeighborSolicit(), Ip6SendRouterSolicit(), and Ip6TimerTicking().

#define IP6_SERVICE_STARTED 1

#define IP6_SERVICE_UNSTARTED 0

Referenced by Ip6CreateService().

#define IP6_STATE_CONFIGED 1

Referenced by EfiIp6Cancel(), EfiIp6Configure(), EfiIp6GetModeData(), EfiIp6Groups(), EfiIp6Neighbors(), EfiIp6Receive(), EfiIp6Routes(), EfiIp6Transmit(), Ip6ConfigProtocol(), Ip6DestroyChildEntryByAddr(), Ip6InstanceEnquePacket(), and Ip6IsValidProtocol().

#define IP6_STATE_UNCONFIGED 0

Referenced by EfiIp6Configure(), EfiIp6Poll(), and Ip6InitProtocol().

Function Documentation

EFI_STATUS EFIAPI EfiIp6Cancel (IN EFI_IP6_PROTOCOL * *This*,
IN EFI_IP6_COMPLETION_TOKEN * *Token*
)

Abort an asynchronous transmit or receive request.

The Cancel() function is used to abort a pending transmit or receive request. If the token is in the transmit or receive request queues, after calling this function, Token->Status will be set to EFI_ABORTED, and then Token->Event will be signaled. If the token is not in one of the queues, which usually means the asynchronous operation has completed, this function will not signal the token, and EFI_NOT_FOUND is returned.

Parameters:

[in]	*This*	The pointer to the EFI_IP6_PROTOCOL instance.
[in]	*Token*	The pointer to a token that has been issued by EFI_IP6_PROTOCOL.Transmit() or EFI_IP6_PROTOCOL.Receive(). If NULL, all pending tokens are aborted. Type EFI_IP6_COMPLETION_TOKEN is defined in EFI_IP6_PROTOCOL.Transmit().

Return values:

EFI_SUCCESS	The asynchronous I/O request was aborted and Token->Event was signaled. When Token is NULL, all pending requests were aborted, and their events were signaled.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This instance has not been started.
EFI_NOT_FOUND	When Token is not NULL, the asynchronous I/O request was not found in the transmit or receive queue. It has either completed or was not issued by Transmit() and Receive().
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

Abort an asynchronous transmit or receive request.

The Cancel() function is used to abort a pending transmit or receive request. If the token is in the transmit or receive request queues, after calling this function, Token->Status will be set to EFI_ABORTED, and then Token->Event will be signaled. If the token is not in one of the queues, which usually means the asynchronous operation has completed, this function will not signal the token, and EFI_NOT_FOUND is returned.

Parameters:

[in]	*This*	Pointer to the EFI_IP6_PROTOCOL instance.
[in]	*Token*	Pointer to a token that has been issued by EFI_IP6_PROTOCOL.Transmit() or EFI_IP6_PROTOCOL.Receive(). If NULL, all pending tokens are aborted. Type EFI_IP6_COMPLETION_TOKEN is defined in EFI_IP6_PROTOCOL.Transmit().

Return values:

EFI_SUCCESS	The asynchronous I/O request was aborted and Token->Event was signaled. When Token is NULL, all pending requests were aborted, and their events were signaled.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This instance has not been started.
EFI_NOT_FOUND	When Token is not NULL, the asynchronous I/O request was not found in the transmit or receive queue. It has either completed or was not issued by Transmit() and Receive().
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References IP6_INSTANCE_FROM_PROTOCOL, IP6_STATE_CONFIGED, Ip6Cancel(), and _IP6_PROTOCOL::State.

EFI_STATUS EFIAPI EfiIp6Configure (IN EFI_IP6_PROTOCOL * *This*,

IN EFI_IP6_CONFIG_DATA * *Ip6ConfigData*
)

Assigns an IPv6 address and subnet mask to this EFI IPv6 Protocol driver instance.

The Configure() function is used to set, change, or reset the operational parameters and filter settings for this EFI IPv6 Protocol instance. Until these parameters have been set, no network traffic can be sent or received by this instance. Once the parameters have been reset (by calling this function with Ip6ConfigData set to NULL), no more traffic can be sent or received until these parameters have been set again. Each EFI IPv6 Protocol instance can be started and stopped independently of each other by enabling or disabling their receive filter settings with the Configure() function.

If Ip6ConfigData.StationAddress is a valid non-zero IPv6 unicast address, it is required to be one of the currently configured IPv6 addresses list in the EFI IPv6 drivers, or else EFI_INVALID_PARAMETER will be returned. If Ip6ConfigData.StationAddress is unspecified, the IPv6 driver will bind a source address according to the source address selection algorithm. Clients could frequently call GetModeData() to check get a currently configured IPv6. If both Ip6ConfigData.StationAddress and Ip6ConfigData.Destination are unspecified, when transmitting the packet afterwards, the source address filled in each outgoing IPv6 packet is decided based on the destination of this packet.

If operational parameters are reset or changed, any pending transmit and receive requests will be cancelled. Their completion token status will be set to EFI_ABORTED, and their events will be signaled.

Parameters:

`[in]`	*This*	The pointer to the EFI_IP6_PROTOCOL instance.
`[in]`	*Ip6ConfigData*	The pointer to the EFI IPv6 Protocol configuration data structure. If NULL, reset the configuration data.

Return values:

EFI_SUCCESS	The driver instance was successfully opened.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Ip6ConfigData.StationAddress is neither zero nor a unicast IPv6 address. ◊ Ip6ConfigData.StationAddress is neither zero nor one of the configured IP addresses in the EFI IPv6 driver. ◊ Ip6ConfigData.DefaultProtocol is illegal.
EFI_OUT_OF_RESOURCES	The EFI IPv6 Protocol driver instance data could not be allocated.
EFI_NO_MAPPING	The IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_ALREADY_STARTED	The interface is already open and must be stopped before the IPv6 address or prefix length can be changed.
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI IPv6 Protocol driver instance was not opened.
EFI_UNSUPPORTED	Default protocol specified through Ip6ConfigData.DefaulProtocol isn't supported.

Assigns an IPv6 address and subnet mask to this EFI IPv6 Protocol driver instance.

The Configure() function is used to set, change, or reset the operational parameters and filter settings for this EFI IPv6 Protocol instance. Until these parameters have been set, no network traffic can be sent or received by this instance. Once the parameters have been reset (by calling this function with Ip6ConfigData set to NULL), no more traffic can be sent or received until these parameters have been set again. Each EFI IPv6 Protocol instance can be started and stopped independently of each other by enabling or disabling their receive

filter settings with the Configure() function.

If Ip6ConfigData.StationAddress is a valid non-zero IPv6 unicast address, it is required to be one of the currently configured IPv6 addresses listed in the EFI IPv6 drivers, or else EFI_INVALID_PARAMETER will be returned. If Ip6ConfigData.StationAddress is unspecified, the IPv6 driver will bind a source address according to the source address selection algorithm. Clients could frequently call GetModeData() to check get currently configured IPv6 address list in the EFI IPv6 driver. If both Ip6ConfigData.StationAddress and Ip6ConfigData.Destination are unspecified, when transmitting the packet afterwards, the source address filled in each outgoing IPv6 packet is decided based on the destination of this packet.

If operational parameters are reset or changed, any pending transmit and receive requests will be cancelled. Their completion token status will be set to EFI_ABORTED and their events will be signaled.

Parameters:

[in]	*This*	Pointer to the EFI_IP6_PROTOCOL instance.
[in]	*Ip6ConfigData*	Pointer to the EFI IPv6 Protocol configuration data structure. If NULL, reset the configuration data.

Return values:

EFI_SUCCESS	The driver instance was successfully opened.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Ip6ConfigData.StationAddress is neither zero nor a unicast IPv6 address. ◊ Ip6ConfigData.StationAddress is neither zero nor one of the configured IP addresses in the EFI IPv6 driver. ◊ Ip6ConfigData.DefaultProtocol is illegal.
EFI_OUT_OF_RESOURCES	The EFI IPv6 Protocol driver instance data could not be allocated.
EFI_NO_MAPPING	The IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_ALREADY_STARTED	The interface is already open and must be stopped before the IPv6 address or prefix length can be changed.
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI IPv6 Protocol driver instance was not opened.
EFI_UNSUPPORTED	Default protocol specified through Ip6ConfigData.DefaulProtocol isn't supported.

References _IP6_PROTOCOL::ConfigData, IP6_INSTANCE_FROM_PROTOCOL, IP6_NO_MAPPING, IP6_STATE_CONFIGED, IP6_STATE_UNCONFIGED, Ip6CleanProtocol(), Ip6ConfigProtocol(), Ip6IsIllegalProtocol(), Ip6IsValidAddress(), Ip6ServiceConfigMnp(), _IP6_SERVICE::LinkLocalDadFail, _IP6_PROTOCOL::Service, and _IP6_PROTOCOL::State.

```
EFI_STATUS EFIAPI
EfiIp6GetModeData
    ( IN EFI_IP6_PROTOCOL *                    This,
      OUT EFI_IP6_MODE_DATA *                  Ip6ModeData,
      OUT EFI_MANAGED_NETWORK_CONFIG_DATA *    MnpConfigData,
      OUT EFI_SIMPLE_NETWORK_MODE *            SnpModeData
    )
```

Gets the current operational settings for this instance of the EFI IPv6 Protocol driver.

The GetModeData() function returns the current operational mode data for this driver instance. The data fields in EFI_IP6_MODE_DATA are read only. This function is used optionally to retrieve the operational mode data of underlying networks or drivers.

Parameters:

[in]	*This*	The pointer to the EFI_IP6_PROTOCOL instance.
[out]	*Ip6ModeData*	The pointer to the EFI IPv6 Protocol mode data structure.
[out]	*MnpConfigData*	The pointer to the managed network configuration data structure.
[out]	*SnpModeData*	The pointer to the simple network mode data structure.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	This is NULL.
EFI_OUT_OF_RESOURCES	The required mode data could not be allocated.

Gets the current operational settings for this instance of the EFI IPv6 Protocol driver.

The GetModeData() function returns the current operational mode data for this driver instance. The data fields in EFI_IP6_MODE_DATA are read only. This function is used optionally to retrieve the operational mode data of underlying networks or drivers.

Parameters:

[in]	*This*	Pointer to the EFI_IP6_PROTOCOL instance.
[out]	*Ip6ModeData*	Pointer to the EFI IPv6 Protocol mode data structure.
[out]	*MnpConfigData*	Pointer to the managed network configuration data structure.
[out]	*SnpModeData*	Pointer to the simple network mode data structure.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	This is NULL.
EFI_OUT_OF_RESOURCES	The required mode data could not be allocated.

References _IP6_PROTOCOL::ConfigData, _IP6_INTERFACE::Configured, _IP6_PROTOCOL::GroupCount, _IP6_PROTOCOL::GroupList, _IP6_PROTOCOL::Interface, IP6_INSTANCE_FROM_PROTOCOL, IP6_STATE_CONFIGED, Ip6BuildEfiAddressList(), Ip6BuildEfiNeighborCache(), Ip6BuildEfiRouteTable(), Ip6BuildPrefixTable(), _IP6_SERVICE::LinkLocalDadFail, _IP6_SERVICE::MaxPacketSize, mIp6SupportedIcmp, _IP6_SERVICE::Mnp, _IP6_SERVICE::RouteTable, _IP6_PROTOCOL::Service, and _IP6_PROTOCOL::State.

```
EFI_STATUS EFIAPI EfiIp6Groups ( IN EFI_IP6_PROTOCOL *   This,
                                 IN BOOLEAN              JoinFlag,
                                 IN EFI_IPv6_ADDRESS *   GroupAddress
                               )
```

Joins and leaves multicast groups.

The Groups() function is used to join and leave multicast group sessions. Joining a group will enable reception of matching multicast packets. Leaving a group will disable reception of matching multicast packets. Source-Specific Multicast isn't required to be supported.

If JoinFlag is FALSE and GroupAddress is NULL, all joined groups will be left.

Parameters:

[in]	*This*	The pointer to the EFI_IP6_PROTOCOL instance.
[in]	*JoinFlag*	Set to TRUE to join the multicast group session and FALSE to leave.
[in]	*GroupAddress*	The pointer to the IPv6 multicast address. This is an optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	One or more of the following is TRUE: ◊ This is NULL. ◊ JoinFlag is TRUE and GroupAddress is NULL. ◊ GroupAddress is not NULL and *GroupAddress is not a multicast IPv6 address. ◊ GroupAddress is not NULL and *GroupAddress is in the range of SSM destination address.
EFI_NOT_STARTED	This instance has not been started.
EFI_OUT_OF_RESOURCES	System resources could not be allocated.
EFI_UNSUPPORTED	This EFI IPv6 Protocol implementation does not support multicast groups.
EFI_ALREADY_STARTED	The group address is already in the group table (when JoinFlag is TRUE).
EFI_NOT_FOUND	The group address is not in the group table (when JoinFlag is FALSE).
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

Joins and leaves multicast groups.

The Groups() function is used to join and leave multicast group sessions. Joining a group will enable reception of matching multicast packets. Leaving a group will disable reception of matching multicast packets. Source-Specific Multicast isn't required to be supported.

If JoinFlag is FALSE and GroupAddress is NULL, all joined groups will be left.

Parameters:

[in]	*This*	Pointer to the EFI_IP6_PROTOCOL instance.
[in]	*JoinFlag*	Set to TRUE to join the multicast group session, and FALSE to leave.
[in]	*GroupAddress*	Pointer to the IPv6 multicast address. This is an optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	One or more of the following is TRUE: ◊ This is NULL. ◊ JoinFlag is TRUE and GroupAddress is NULL. ◊ GroupAddress is not NULL and *GroupAddress is not a multicast IPv6 address. ◊ GroupAddress is not NULL and *GroupAddress is in the range of SSM destination address.
EFI_NOT_STARTED	This instance has not been started.
EFI_OUT_OF_RESOURCES	System resources could not be allocated.
EFI_UNSUPPORTED	This EFI IPv6 Protocol implementation does not support multicast groups.
EFI_ALREADY_STARTED	

	The group address is already in the group table (when JoinFlag is TRUE).
EFI_NOT_FOUND	The group address is not in the group table (when JoinFlag is FALSE).
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References IP6_INSTANCE_FROM_PROTOCOL, IP6_STATE_CONFIGED, Ip6Groups(), _IP6_SERVICE::LinkLocalDadFail, _IP6_PROTOCOL::Service, and _IP6_PROTOCOL::State.

```
EFI_STATUS EFIAPI EfiIp6Neighbors ( IN EFI_IP6_PROTOCOL *   This,
                                    IN BOOLEAN              DeleteFlag,
                                    IN EFI_IPv6_ADDRESS *   TargetIp6Address,
                                    IN EFI_MAC_ADDRESS *    TargetLinkAddress,
                                    IN UINT32               Timeout,
                                    IN BOOLEAN              Override
                                  )
```

Add or delete Neighbor cache entries.

The Neighbors() function is used to add, update, or delete an entry from a neighbor cache. IPv6 neighbor cache entries are typically inserted and updated by the network protocol driver as network traffic is processed. Most neighbor cache entries will timeout and be deleted if the network traffic stops. Neighbor cache entries that were inserted by Neighbors() may be static (will not timeout) or dynamic (will timeout).

The implementation should follow the neighbor cache timeout mechanism defined in RFC4861. The default neighbor cache timeout value should be tuned for the expected network environment.

Parameters:

`[in]` *This*	The pointer to the EFI_IP6_PROTOCOL instance.
`[in]` *DeleteFlag*	Set to TRUE to delete the specified cache entry. Set to FALSE to add (or update, if it already exists and Override is TRUE) the specified cache entry. TargetIp6Address is used as the key to find the requested cache entry.
`[in]` *TargetIp6Address*	The pointer to the Target IPv6 address.
`[in]` *TargetLinkAddress*	The pointer to link-layer address of the target. Ignored if NULL.
`[in]` *Timeout*	Time in 100-ns units that this entry will remain in the neighbor cache, it will be deleted after Timeout. A value of zero means that the entry is permanent. A non-zero value means that the entry is dynamic.
`[in]` *Override*	If TRUE, the cached link-layer address of the matching entry will be overridden and updated; if FALSE, EFI_ACCESS_DENIED will be returned if a corresponding cache entry already exists.

Return values:

EFI_SUCCESS	The data has been queued for transmission.
EFI_NOT_STARTED	This instance has not been started.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ TargetIpAddress is NULL. ◊ *TargetLinkAddress is invalid when not NULL. ◊ *TargetIpAddress is not a valid unicast IPv6 address. ◊ *TargetIpAddress is one of the local configured IPv6 addresses.

EFI_OUT_OF_RESOURCES	Could not add the entry to the neighbor cache.
EFI_NOT_FOUND	This entry is not in the neighbor cache (when DeleteFlag is TRUE or when DeleteFlag is FALSE while TargetLinkAddress is NULL.).
EFI_ACCESS_DENIED	The to-be-added entry is already defined in the neighbor cache, and that entry is tagged as un-overridden (when Override is FALSE).

Add or delete Neighbor cache entries.

The Neighbors() function is used to add, update, or delete an entry from neighbor cache. IPv6 neighbor cache entries are typically inserted and updated by the network protocol driver as network traffic is processed. Most neighbor cache entries will timeout and be deleted if the network traffic stops. Neighbor cache entries that were inserted by Neighbors() may be static (will not timeout) or dynamic (will timeout).

The implementation should follow the neighbor cache timeout mechanism which is defined in RFC4861. The default neighbor cache timeout value should be tuned for the expected network environment

Parameters:

`[in]` *This*	Pointer to the EFI_IP6_PROTOCOL instance.
`[in]` *DeleteFlag*	Set to TRUE to delete the specified cache entry, set to FALSE to add (or update, if it already exists and Override is TRUE) the specified cache entry. TargetIp6Address is used as the key to find the requested cache entry.
`[in]` *TargetIp6Address*	Pointer to the Target IPv6 address.
`[in]` *TargetLinkAddress*	Pointer to the link-layer address of the target. Ignored if NULL.
`[in]` *Timeout*	Time in 100-ns units that this entry will remain in the neighbor cache, it will be deleted after Timeout. A value of zero means that the entry is permanent. A non-zero value means that the entry is dynamic.
`[in]` *Override*	If TRUE, the cached link-layer address of the matching entry will be overridden and updated; if FALSE, EFI_ACCESS_DENIED will be returned if a corresponding cache entry already existed.

Return values:

EFI_SUCCESS	The data has been queued for transmission.
EFI_NOT_STARTED	This instance has not been started.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ TargetIpAddress is NULL. ◊ *TargetLinkAddress is invalid when not NULL. ◊ *TargetIpAddress is not a valid unicast IPv6 address. ◊ *TargetIpAddress is one of the local configured IPv6 addresses.
EFI_OUT_OF_RESOURCES	Could not add the entry to the neighbor cache.
EFI_NOT_FOUND	This entry is not in the neighbor cache (when DeleteFlag is TRUE or when DeleteFlag is FALSE while TargetLinkAddress is NULL.).
EFI_ACCESS_DENIED	The to-be-added entry is already defined in the neighbor cache, and that entry is tagged as un-overridden (when Override is FALSE).

References IP6_INSTANCE_FROM_PROTOCOL, IP6_STATE_CONFIGED, Ip6AddNeighbor(), Ip6DelNeighbor(), Ip6IsOneOfSetAddress(), Ip6IsValidAddress(), Ip6IsValidLinkAddress(), _IP6_SERVICE::LinkLocalDadFail, _IP6_PROTOCOL::Service, and _IP6_PROTOCOL::State.

EFI_STATUS EFIAPI EfiIp6Poll (IN EFI_IP6_PROTOCOL * *This*)

Polls for incoming data packets and processes outgoing data packets.

The Poll() function polls for incoming data packets and processes outgoing data packets. Network drivers and applications can call the EFI_IP6_PROTOCOL.Poll() function to increase the rate that data packets are moved between the communications device and the transmit and receive queues.

In some systems the periodic timer event may not poll the underlying communications device fast enough to transmit and/or receive all data packets without missing incoming packets or dropping outgoing packets. Drivers and applications that are experiencing packet loss should try calling the EFI_IP6_PROTOCOL.Poll() function more often.

Parameters:

`[in]` *This* The pointer to the EFI_IP6_PROTOCOL instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_NOT_STARTED	This EFI IPv6 Protocol instance has not been started.
EFI_INVALID_PARAMETER	This is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_NOT_READY	No incoming or outgoing data was processed.
EFI_TIMEOUT	Data was dropped out of the transmit and/or receive queue. Consider increasing the polling rate.

Polls for incoming data packets, and processes outgoing data packets.

The Poll() function polls for incoming data packets and processes outgoing data packets. Network drivers and applications can call the EFI_IP6_PROTOCOL.Poll() function to increase the rate that data packets are moved between the communications device and the transmit and receive queues.

In some systems the periodic timer event may not poll the underlying communications device fast enough to transmit and/or receive all data packets without missing incoming packets or dropping outgoing packets. Drivers and applications that are experiencing packet loss should try calling the EFI_IP6_PROTOCOL.Poll() function more often.

Parameters:

`[in]` *This* Pointer to the EFI_IP6_PROTOCOL instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_NOT_STARTED	This EFI IPv6 Protocol instance has not been started.
EFI_INVALID_PARAMETER	This is NULL.
EFI_DEVICE_ERROR	An unexpected system error or network error occurred.
EFI_NOT_READY	No incoming or outgoing data was processed.
EFI_TIMEOUT	Data was dropped out of the transmit and/or receive queue. Consider increasing the polling rate.

References IP6_INSTANCE_FROM_PROTOCOL, IP6_STATE_UNCONFIGED, _IP6_SERVICE::LinkLocalDadFail, _IP6_SERVICE::Mnp, _IP6_PROTOCOL::Service, and _IP6_PROTOCOL::State.

```
EFI_STATUS EFIAPI EfiIp6Receive ( IN EFI_IP6_PROTOCOL *              This,
                                  IN EFI_IP6_COMPLETION_TOKEN *      Token
```

)

Places a receiving request into the receiving queue.

The Receive() function places a completion token into the receive packet queue. This function is always asynchronous.

The Token.Event field in the completion token must be filled in by the caller and cannot be NULL. When the receive operation completes, the EFI IPv6 Protocol driver updates the Token.Status and Token.Packet.RxData fields and the Token.Event is signaled.

Current Udp implementation creates an IP child for each Udp child. It initates a asynchronous receive immediately whether or not there is no mapping. Therefore, disable the returning EFI_NO_MAPPING for now. To enable it, the following check must be performed:

if (NetIp6IsUnspecifiedAddr (&Config->StationAddress) && IP6_NO_MAPPING (IpInstance)) { Status = EFI_NO_MAPPING; goto Exit; }

Parameters:

`[in]` *This* The pointer to the EFI_IP6_PROTOCOL instance.

`[in]` *Token* The pointer to a token that is associated with the receive data descriptor.

Return values:

EFI_SUCCESS	The receive completion token was cached.
EFI_NOT_STARTED	This EFI IPv6 Protocol instance has not been started.
EFI_NO_MAPPING	When IP6 driver responsible for binding source address to this instance, while no source address is available for use.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token.Event is NULL.
EFI_OUT_OF_RESOURCES	The receive completion token could not be queued due to a lack of system resources (usually memory).
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI IPv6 Protocol instance has been reset to startup defaults.
EFI_ACCESS_DENIED	The receive completion token with the same Token.Event was already in the receive queue.
EFI_NOT_READY	The receive request could not be queued because the receive queue is full.

Places a receiving request into the receiving queue.

The Receive() function places a completion token into the receive packet queue. This function is always asynchronous.

The Token.Event field in the completion token must be filled in by the caller and cannot be NULL. When the receive operation completes, the EFI IPv6 Protocol driver updates the Token.Status and Token.Packet.RxData fields and the Token.Event is signaled.

Current Udp implementation creates an IP child for each Udp child. It initates a asynchronous receive immediately no matter whether there is no mapping or not. Therefore, disable the returning EFI_NO_MAPPING for now. To enable it, the following check must be performed:

if (NetIp6IsUnspecifiedAddr (&Config->StationAddress) && IP6_NO_MAPPING (IpInstance)) { Status =

EFI_NO_MAPPING; goto Exit; }

Parameters:

[in]	*This*	Pointer to the EFI_IP6_PROTOCOL instance.
[in]	*Token*	Pointer to a token that is associated with the receive data descriptor.

Return values:

EFI_SUCCESS	The receive completion token was cached.
EFI_NOT_STARTED	This EFI IPv6 Protocol instance has not been started.
EFI_NO_MAPPING	When IP6 driver responsible for binding source address to this instance, while no source address is available for use.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token.Event is NULL.
EFI_OUT_OF_RESOURCES	The receive completion token could not be queued due to a lack of system resources (usually memory).
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI IPv6 Protocol instance has been reset to startup defaults.
EFI_ACCESS_DENIED	The receive completion token with the same Token.Event was already in the receive queue.
EFI_NOT_READY	The receive request could not be queued because the receive queue is full.

References IP6_INSTANCE_FROM_PROTOCOL, IP6_STATE_CONFIGED, Ip6InstanceDeliverPacket(), Ip6TokenExist(), _IP6_SERVICE::LinkLocalDadFail, _IP6_PROTOCOL::RxTokens, _IP6_PROTOCOL::Service, and _IP6_PROTOCOL::State.

```
EFI_STATUS EFIAPI EfiIp6Routes ( IN EFI_IP6_PROTOCOL *  This,
                                 IN BOOLEAN             DeleteRoute,
                                 IN EFI_IPv6_ADDRESS *  Destination,
                                 IN UINT8               PrefixLength,
                                 IN EFI_IPv6_ADDRESS *  GatewayAddress
                               )
```

Adds and deletes routing table entries.

The Routes() function adds a route to or deletes a route from the routing table.

Routes are determined by comparing the leftmost PrefixLength bits of Destination with the destination IPv6 address arithmetically. The gateway address must be on the same subnet as the configured station address.

The default route is added with Destination and PrefixLegth both set to all zeros. The default route matches all destination IPv6 addresses that do not match any other routes.

All EFI IPv6 Protocol instances share a routing table.

Parameters:

[in]	*This*	The pointer to the EFI_IP6_PROTOCOL instance.
[in]	*DeleteRoute*	Set to TRUE to delete this route from the routing table. Set to FALSE to add this route to the routing table. Destination, PrefixLength and Gateway are used as the key to each route entry.

[in]	*Destination*	The address prefix of the subnet that needs to be routed. This is an optional parameter that may be NULL.
[in]	*PrefixLength*	The prefix length of Destination. Ignored if Destination is NULL.
[in]	*GatewayAddress*	The unicast gateway IPv6 address for this route. This is an optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_STARTED	The driver instance has not been started.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ When DeleteRoute is TRUE, both Destination and GatewayAddress are NULL. ◊ When DeleteRoute is FALSE, either Destination or GatewayAddress is NULL. ◊ *GatewayAddress is not a valid unicast IPv6 address. ◊ *GatewayAddress is one of the local configured IPv6 addresses.
EFI_OUT_OF_RESOURCES	Could not add the entry to the routing table.
EFI_NOT_FOUND	This route is not in the routing table (when DeleteRoute is TRUE).
EFI_ACCESS_DENIED	The route is already defined in the routing table (when DeleteRoute is FALSE).

Adds and deletes routing table entries.

The Routes() function adds a route to, or deletes a route from, the routing table.

Routes are determined by comparing the leftmost PrefixLength bits of Destination with the destination IPv6 address arithmetically. The gateway address must be on the same subnet as the configured station address.

The default route is added with Destination and PrefixLegth both set to all zeros. The default route matches all destination IPv6 addresses that do not match any other routes.

All EFI IPv6 Protocol instances share a routing table.

Parameters:

[in]	*This*	Pointer to the EFI_IP6_PROTOCOL instance.
[in]	*DeleteRoute*	Set to TRUE to delete this route from the routing table. Set to FALSE to add this route to the routing table. Destination, PrefixLength and Gateway are used as the key to each route entry.
[in]	*Destination*	The address prefix of the subnet that needs to be routed. This is an optional parameter that may be NULL.
[in]	*PrefixLength*	The prefix length of Destination. Ignored if Destination is NULL.
[in]	*GatewayAddress*	The unicast gateway IPv6 address for this route. This is an optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_STARTED	The driver instance has not been started.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ When DeleteRoute is TRUE, both Destination and GatewayAddress are NULL.

◊ When DeleteRoute is FALSE, either Destination or GatewayAddress is NULL.
◊ *GatewayAddress is not a valid unicast IPv6 address.
◊ *GatewayAddress is one of the local configured IPv6 addresses.

EFI_OUT_OF_RESOURCES	Could not add the entry to the routing table.
EFI_NOT_FOUND	This route is not in the routing table (when DeleteRoute is TRUE).
EFI_ACCESS_DENIED	The route is already defined in the routing table (when DeleteRoute is FALSE).

References _IP6_PROTOCOL::ConfigData, IP6_INSTANCE_FROM_PROTOCOL, IP6_STATE_CONFIGED, Ip6AddRoute(), Ip6DelRoute(), Ip6IsValidAddress(), _IP6_SERVICE::LinkLocalDadFail, _IP6_SERVICE::RouteTable, _IP6_PROTOCOL::Service, and _IP6_PROTOCOL::State.

```
EFI_STATUS EFIAPI EfiIp6Transmit ( IN EFI_IP6_PROTOCOL *              This,
                                   IN EFI_IP6_COMPLETION_TOKEN *      Token
                                 )
```

Places outgoing data packets into the transmit queue.

The Transmit() function places a sending request in the transmit queue of this EFI IPv6 Protocol instance. Whenever the packet in the token is sent out or some errors occur, the event in the token will be signaled and the status is updated.

Parameters:

[in]	*This*	The pointer to the EFI_IP6_PROTOCOL instance.
[in]	*Token*	The pointer to the transmit token.

Return values:

EFI_SUCCESS	The data has been queued for transmission.
EFI_NOT_STARTED	This instance has not been started.
EFI_NO_MAPPING	The IPv6 driver was responsible for choosing a source address for this transmission, but no source address was available for use.
EFI_INVALID_PARAMETER	One or more of the following is TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token.Event is NULL. ◊ Token.Packet.TxData is NULL. ◊ Token.Packet.ExtHdrsLength is not zero and Token.Packet.ExtHdrs is NULL. ◊ Token.Packet.FragmentCount is zero. ◊ One or more of the Token.Packet.TxData. FragmentTable[].FragmentLength fields is zero. ◊ One or more of the Token.Packet.TxData. FragmentTable[].FragmentBuffer fields is NULL. ◊ Token.Packet.TxData.DataLength is zero or not equal to the sum of fragment lengths. ◊ Token.Packet.TxData.DestinationAddress is non- zero when DestinationAddress is configured as non-zero when doing Configure() for this EFI IPv6 protocol instance. ◊ Token.Packet.TxData.DestinationAddress is unspecified when DestinationAddress is unspecified when doing

Configure() for this EFI IPv6 protocol instance.

EFI_ACCESS_DENIED	The transmit completion token with the same Token. The event was already in the transmit queue.
EFI_NOT_READY	The completion token could not be queued because the transmit queue is full.
EFI_NOT_FOUND	Not route is found to the destination address.
EFI_OUT_OF_RESOURCES	Could not queue the transmit data.
EFI_BUFFER_TOO_SMALL	Token.Packet.TxData.TotalDataLength is too short to transmit.
EFI_BAD_BUFFER_SIZE	If Token.Packet.TxData.DataLength is beyond the maximum that which can be described through the Fragment Offset field in Fragment header when performing fragmentation.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

Places outgoing data packets into the transmit queue.

The Transmit() function places a sending request in the transmit queue of this EFI IPv6 Protocol instance. Whenever the packet in the token is sent out or some errors occur, the event in the token will be signaled, and the status is updated.

Parameters:

`[in]` *This* Pointer to the EFI_IP6_PROTOCOL instance.

`[in]` *Token* Pointer to the transmit token.

Return values:

EFI_SUCCESS	The data has been queued for transmission.
EFI_NOT_STARTED	This instance has not been started.
EFI_NO_MAPPING	The IPv6 driver was responsible for choosing a source address for this transmission, but no source address was available for use.
EFI_INVALID_PARAMETER	One or more of the following is TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token.Event is NULL. ◊ Token.Packet.TxData is NULL. ◊ Token.Packet.ExtHdrsLength is not zero and Token.Packet.ExtHdrs is NULL. ◊ Token.Packet.FragmentCount is zero. ◊ One or more of the Token.Packet.TxData. FragmentTable[].FragmentLength fields is zero. ◊ One or more of the Token.Packet.TxData. FragmentTable[].FragmentBuffer fields is NULL. ◊ Token.Packet.TxData.DataLength is zero or not equal to the sum of fragment lengths. ◊ Token.Packet.TxData.DestinationAddress is non zero when DestinationAddress is configured as non-zero when doing Configure() for this EFI IPv6 protocol instance. ◊ Token.Packet.TxData.DestinationAddress is unspecified when DestinationAddress is unspecified when doing Configure() for this EFI IPv6 protocol instance.
EFI_ACCESS_DENIED	The transmit completion token with the same Token. Event was already in the transmit queue.
EFI_NOT_READY	The completion token could not be queued because the transmit queue is full.
EFI_NOT_FOUND	Not route is found to destination address.

EFI_OUT_OF_RESOURCES	Could not queue the transmit data.
EFI_BUFFER_TOO_SMALL	Token.Packet.TxData.TotalDataLength is too short to transmit.
EFI_BAD_BUFFER_SIZE	If Token.Packet.TxData.DataLength is beyond the maximum that which can be described through the Fragment Offset field in Fragment header when performing fragmentation.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References _IP6_PROTOCOL::ConfigData, IP6_INSTANCE_FROM_PROTOCOL, IP6_MAX_HEADLEN, IP6_STATE_CONFIGED, IP6_US_TO_SEC, Ip6FreeTxToken(), Ip6OnPacketSent(), Ip6Output(), Ip6SelectSourceAddress(), Ip6TokenExist(), Ip6TxTokenValid(), IP6_TXTOKEN_WRAP::IpInstance, IP6_TXTOKEN_WRAP::Life, _IP6_SERVICE::LinkLocalDadFail, IP6_TXTOKEN_WRAP::Packet, IP6_TXTOKEN_WRAP::Sent, _IP6_PROTOCOL::Service, _IP6_PROTOCOL::State, IP6_TXTOKEN_WRAP::Token, and _IP6_PROTOCOL::TxTokens.

EFI_STATUS Ip6Cancel (IN IP6_PROTOCOL * *IpInstance*,
IN EFI_IP6_COMPLETION_TOKEN * *Token*
)

Cancel the user's receive/transmit request. It is the worker function of EfiIp6Cancel API.

Parameters:

[in]	*IpInstance*	The IP6 child.
[in]	*Token*	The token to cancel. If NULL, all tokens will be cancelled.

Return values:

EFI_SUCCESS	The token was cancelled.
EFI_NOT_FOUND	The token isn't found on either the transmit or receive queue.
EFI_DEVICE_ERROR	Not all tokens are cancelled when Token is NULL.

Cancel the user's receive/transmit request. It is the worker function of EfiIp6Cancel API.

Parameters:

[in]	*IpInstance*	The IP6 child.
[in]	*Token*	The token to cancel. If NULL, all token will be cancelled.

Return values:

EFI_SUCCESS	The token is cancelled.
EFI_NOT_FOUND	The token isn't found on either the transmit/receive queue.
EFI_DEVICE_ERROR	Not all tokens are cancelled when Token is NULL.

References Ip6CancelRxTokens(), and Ip6CancelTxTokens().

Referenced by EfiIp6Cancel(), Ip6CleanProtocol(), and Ip6ConfigProtocol().

EFI_STATUS Ip6CleanProtocol (IN OUT IP6_PROTOCOL * *IpInstance*)

Clean up the IP6 child, release all the resources used by it.

Parameters:

[in,out]	*IpInstance*	The IP6 child to clean up.

Return values:

EFI_SUCCESS	The IP6 child was cleaned up

EFI_DEVICE_ERROR Some resources failed to be released.

Clean up the IP6 child, and release all the resources used by it.

Parameters:

`[in,out]` *IpInstance* The IP6 child to clean up.

Return values:

EFI_SUCCESS The IP6 child is cleaned up.
EFI_DEVICE_ERROR Some resources failed to be released.

References Ip6Cancel(), Ip6CleanInterface(), and Ip6Groups().

Referenced by EfiIp6Configure(), Ip6ServiceBindingCreateChild(), and Ip6ServiceBindingDestroyChild().

VOID EFIAPI Ip6FreeTxToken (IN VOID * *Context*)

The callback function for the net buffer which wraps the user's transmit token. Although this function seems simple, there are some subtle aspects. When a user requests the IP to transmit a packet by passing it a token, the token is wrapped in an IP6_TXTOKEN_WRAP and the data is wrapped in a net buffer. The net buffer's Free function is set to Ip6FreeTxToken. The Token and token wrap are added to the IP child's TxToken map. Then the buffer is passed to Ip6Output for transmission. If an error occurs before that, the buffer is freed, which in turn frees the token wrap. The wrap may have been added to the TxToken map or not, and the user's event shouldn't be signaled because we are still in the EfiIp6Transmit. If the buffer has been sent by Ip6Output, it should be removed from the TxToken map and the user's event signaled. The token wrap and buffer are bound together. Refer to the comments in Ip6Output for information about IP fragmentation.

Parameters:

`[in]` *Context* The token's wrap.

The callback function for the net buffer which wraps the user's transmit token. Although this function seems simple, there are some subtle aspects. When user requests the IP to transmit a packet by passing it a token, the token is wrapped in an IP6_TXTOKEN_WRAP and the data is wrapped in an net buffer. The net buffer's Free function is set to Ip6FreeTxToken. The Token and token wrap are added to the IP child's TxToken map. Then the buffer is passed to Ip6Output for transmission. If an error happened before that, the buffer is freed, which in turn frees the token wrap. The wrap may have been added to the TxToken map or not, and the user's event shouldn't be fired because we are still in the EfiIp6Transmit. If the buffer has been sent by Ip6Output, it should be removed from the TxToken map and user's event signaled. The token wrap and buffer are bound together. Check the comments in Ip6Output for information about IP fragmentation.

Parameters:

`[in]` *Context* The token's wrap.

References IP6_TXTOKEN_WRAP::IpInstance, IP6_TXTOKEN_WRAP::IpSecRecycleSignal, IP6_TXTOKEN_WRAP::Sent, IP6_TXTOKEN_WRAP::Token, and _IP6_PROTOCOL::TxTokens.

Referenced by EfiIp6Transmit(), and Ip6IpSecProcessPacket().

VOID Ip6InitProtocol (IN IP6_SERVICE * *IpSb*,
IN OUT IP6_PROTOCOL * *IpInstance*
)

Initialize the IP6_PROTOCOL structure to the unconfigured states.

Parameters:

[in]	*IpSb*	The IP6 service instance.
[in,out]	*IpInstance*	The IP6 child instance.

Intiialize the IP6_PROTOCOL structure to the unconfigured states.

Parameters:

[in]	*IpSb*	The IP6 service instance.
[in,out]	*IpInstance*	The IP6 child instance.

References IP6_PROTOCOL_SIGNATURE, IP6_STATE_UNCONFIGED, and mEfiIp6ProtocolTemplete.

Referenced by Ip6ServiceBindingCreateChild().

```
EFI_STATUS Ip6ServiceConfigMnp ( IN IP6_SERVICE *   IpSb,
                                 IN BOOLEAN         Force
                               )
```

Config the MNP parameter used by IP. The IP driver use one MNP child to transmit/receive frames. By default, it configures MNP to receive unicast/multicast/broadcast. And it will enable/disable the promiscuous receive according to whether there is IP child enable that or not. If Force is FALSE, it will iterate through all the IP children to check whether the promiscuous receive setting has been changed. If it hasn't been changed, it won't reconfigure the MNP. If Force is TRUE, the MNP is configured whether that is changed or not.

Parameters:

[in]	*IpSb*	The IP6 service instance that is to be changed.
[in]	*Force*	Force the configuration or not.

Return values:

EFI_SUCCESS	The MNP successfully configured/reconfigured.
Others	The configuration failed.

Configure the MNP parameter used by IP. The IP driver uses one MNP child to transmit/receive frames. By default, it configures MNP to receive unicast/multicast/broadcast. Also, it will enable/disable the promiscuous receive according to whether there is IP child enable that or not. If Force is FALSE, it will iterate through all the IP children to check whether the promiscuous receive setting has been changed. If it hasn't been changed, it won't reconfigure the MNP. If Force is TRUE, the MNP is configured whether that is changed or not.

Parameters:

[in]	*IpSb*	The IP6 service instance that is to be changed.
[in]	*Force*	Force the configuration or not.

Return values:

EFI_SUCCESS	The MNP successfully configured/reconfigured.
Others	Configuration failed.

References _IP6_PROTOCOL::ConfigData, _IP6_INTERFACE::IpInstances, and _IP6_INTERFACE::PromiscRecv.

Referenced by EfiIp6Configure(), and Ip6CreateService().

Variable Documentation

EFI_IPSEC2_PROTOCOL* mIpSec

Referenced by InitializeIpSecConfig(), and Ip6IpSecProcessPacket().

BOOLEAN mIpSec2Installed

Referenced by Ip6IpSecProcessPacket(), and IpSec2InstalledCallback().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Ip6Dxe/Ip6Input.c File Reference

Functions

IP6_ASSEMBLE_ENTRY * Ip6CreateAssembleEntry (IN EFI_IPv6_ADDRESS *Dst, IN EFI_IPv6_ADDRESS *Src, IN UINT32 Id)
VOID Ip6FreeAssembleEntry (IN IP6_ASSEMBLE_ENTRY *Assemble)
VOID EFIAPI Ip6OnFreeFragments (IN VOID *Arg)
VOID Ip6TrimPacket (IN OUT NET_BUF *Packet, IN INTN Start, IN INTN End)
NET_BUF * Ip6Reassemble (IN OUT IP6_ASSEMBLE_TABLE *Table, IN NET_BUF *Packet)
VOID EFIAPI Ip6IpSecFree (IN VOID *Arg)
EFI_STATUS Ip6IpSecProcessPacket (IN IP6_SERVICE *IpSb, IN OUT EFI_IP6_HEADER **Head, IN OUT UINT8 *LastHead, IN OUT NET_BUF **Netbuf, IN OUT UINT8 **ExtHdrs, IN OUT UINT32 *ExtHdrsLen, IN EFI_IPSEC_TRAFFIC_DIR Direction, IN VOID *Context)
EFI_STATUS Ip6PreProcessPacket (IN IP6_SERVICE *IpSb, IN OUT NET_BUF **Packet, IN UINT32 Flag, OUT UINT8 **Payload, OUT UINT8 **LastHead, OUT UINT32 *ExtHdrsLen, OUT UINT32 *UnFragmentLen, OUT BOOLEAN *Fragmented, OUT EFI_IP6_HEADER **Head)
VOID Ip6AcceptFrame (IN NET_BUF *Packet, IN EFI_STATUS IoStatus, IN UINT32 Flag, IN VOID *Context)
VOID Ip6CreateAssembleTable (IN OUT IP6_ASSEMBLE_TABLE *Table)
VOID Ip6CleanAssembleTable (IN OUT IP6_ASSEMBLE_TABLE *Table)
VOID EFIAPI Ip6OnRecyclePacket (IN EFI_EVENT Event, IN VOID *Context)
IP6_RXDATA_WRAP * Ip6WrapRxData (IN IP6_PROTOCOL *IpInstance, IN NET_BUF *Packet)
BOOLEAN Ip6InstanceFrameAcceptable (IN IP6_PROTOCOL *IpInstance, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
EFI_STATUS Ip6InstanceEnquePacket (IN IP6_PROTOCOL *IpInstance, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
EFI_STATUS Ip6InstanceDeliverPacket (IN IP6_PROTOCOL *IpInstance)
INTN Ip6InterfaceEnquePacket (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet, IN IP6_INTERFACE *IpIf)
VOID Ip6InterfaceDeliverPacket (IN IP6_SERVICE *IpSb, IN IP6_INTERFACE *IpIf)
EFI_STATUS Ip6Demultiplex (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
EFI_STATUS EFIAPI Ip6SentPacketTicking (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Context)
VOID Ip6PacketTimerTicking (IN IP6_SERVICE *IpSb)

Detailed Description

IP6 internal functions to process the incoming packets.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.
(C) Copyright 2015 Hewlett-Packard Development Company, L.P.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at

http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
VOID Ip6AcceptFrame ( IN NET_BUF *      Packet,
                      IN EFI_STATUS     IoStatus,
                      IN UINT32         Flag,
                      IN VOID *         Context
                    )
```

The IP6 input routine. It is called by the IP6_INTERFACE when an IP6 fragment is received from MNP.

Parameters:

- [in] *Packet* The IP6 packet received.
- [in] *IoStatus* The return status of receive request.
- [in] *Flag* The link layer flag for the packet received, such as multicast.
- [in] *Context* The IP6 service instance that owns the MNP.

References IP6_GET_CLIP_INFO, IP6_SERVICE_DESTROY, IP6_SERVICE_SIGNATURE, Ip6AcceptFrame(), Ip6Demultiplex(), Ip6IcmpHandle(), Ip6IpSecProcessPacket(), Ip6PreProcessPacket(), Ip6ReceiveFrame(), and _IP6_SERVICE::State.

Referenced by Ip6AcceptFrame(), and Ip6DriverBindingStart().

```
VOID Ip6CleanAssembleTable ( IN OUT IP6_ASSEMBLE_TABLE *   Table   )
```

Clean up the assemble table by removing all of the fragments and assemble entries.

Parameters:

- [in,out] *Table* The assemble table to clean up.

References IP6_ASSEMLE_HASH_SIZE, and Ip6FreeAssembleEntry().

Referenced by Ip6CleanService().

```
IP6_ASSEMBLE_ENTRY* Ip6CreateAssembleEntry ( IN EFI_IPv6_ADDRESS *   Dst,
                                             IN EFI_IPv6_ADDRESS *   Src,
                                             IN UINT32               Id
                                           )
```

Create an empty assemble entry for the packet identified by (Dst, Src, Id). The default life for the packet is 60 seconds.

Parameters:

- [in] *Dst* The destination address.
- [in] *Src* The source address.
- [in] *Id* The ID field in the IP header.

Returns:

NULL if failed to allocate memory for the entry. Otherwise, the pointer to the just created reassemble entry.

References IP6_ASSEMBLE_ENTRY::CurLen, IP6_ASSEMBLE_ENTRY::Dst, IP6_ASSEMBLE_ENTRY::Fragments, IP6_ASSEMBLE_ENTRY::Head, IP6_ASSEMBLE_ENTRY::Id, IP6_ASSEMBLE_ENTRY::Info, IP6_FRAGMENT_LIFE, IP6_ASSEMBLE_ENTRY::Life, IP6_ASSEMBLE_ENTRY::Packet, IP6_ASSEMBLE_ENTRY::Src, and IP6_ASSEMBLE_ENTRY::TotalLen.

Referenced by Ip6Reassemble().

VOID Ip6CreateAssembleTable (IN OUT IP6_ASSEMBLE_TABLE * *Table*)

Initialize an already allocated assemble table. This is generally the assemble table embedded in the IP6 service instance.

Parameters:

`[in,out]` *Table* The assemble table to initialize.

References IP6_ASSEMLE_HASH_SIZE.

Referenced by Ip6CreateService().

EFI_STATUS Ip6Demultiplex (IN IP6_SERVICE * *IpSb*,
IN EFI_IP6_HEADER * *Head*,
IN NET_BUF * *Packet*
)

De-multiplex the packet. the packet delivery is processed in two passes. The first pass will enqueue a shared copy of the packet to each IP6 child that accepts the packet. The second pass will deliver a non-shared copy of the packet to each IP6 child that has pending receive requests. Data is copied if more than one child wants to consume the packet, because each IP child needs its own copy of the packet to make changes.

Parameters:

`[in]` *IpSb* The IP6 service instance that received the packet.
`[in]` *Head* The header of the received packet.
`[in]` *Packet* The data of the received packet.

Return values:

EFI_NOT_FOUND No IP child accepts the packet.
EFI_SUCCESS The packet is enqueued or delivered to some IP children.

References _IP6_INTERFACE::Configured, Ip6InterfaceDeliverPacket(), and Ip6InterfaceEnquePacket().

Referenced by Ip6AcceptFrame(), Ip6ProcessIcmpError(), and Ip6ProcessIcmpInformation().

VOID Ip6FreeAssembleEntry (IN IP6_ASSEMBLE_ENTRY * *Assemble*)

Release all the fragments of a packet, then free the assemble entry.

Parameters:

`[in]` *Assemble* The assemble entry to free.

Referenced by Ip6CleanAssembleTable(), Ip6OnFreeFragments(), Ip6PacketTimerTicking(), and Ip6Reassemble().

EFI_STATUS Ip6InstanceDeliverPacket (IN IP6_PROTOCOL * *IpInstance*)

Deliver the received packets to the upper layer if there are both received requests and enqueued packets. If the enqueued packet is shared, it will duplicate it to a non-shared packet, release the shared packet, then deliver the non-shared packet up.

Parameters:

`[in]` *IpInstance* The IP child to deliver the packet up.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate resources to deliver the packets.
EFI_SUCCESS	All the enqueued packets that can be delivered are delivered up.

References IP6_GET_CLIP_INFO, Ip6WrapRxData(), IP6_RXDATA_WRAP::Link, and IP6_RXDATA_WRAP::RxData.

Referenced by EfiIp6Receive(), and Ip6InterfaceDeliverPacket().

EFI_STATUS Ip6InstanceEnquePacket (IN IP6_PROTOCOL * *IpInstance*,
IN EFI_IP6_HEADER * *Head*,
IN NET_BUF * *Packet*
)

Enqueue a shared copy of the packet to the IP6 child if the packet is acceptable to it. Here the data of the packet is shared, but the net buffer isn't.

Parameters:

IpInstance	The IP6 child to enqueue the packet to.
Head	The IP header of the received packet.
Packet	The data of the received packet.

Return values:

EFI_NOT_STARTED	The IP child hasn't been configured.
EFI_INVALID_PARAMETER	The child doesn't want to receive the packet.
EFI_OUT_OF_RESOURCES	Failed to allocate some resources
EFI_SUCCESS	A shared copy the packet is enqueued to the child.

References IP6_GET_CLIP_INFO, IP6_STATE_CONFIGED, IP6_US_TO_SEC, Ip6InstanceFrameAcceptable(), and IP6_CLIP_INFO::Life.

Referenced by Ip6InterfaceEnquePacket().

BOOLEAN Ip6InstanceFrameAcceptable (IN IP6_PROTOCOL * *IpInstance*,
IN EFI_IP6_HEADER * *Head*,
IN NET_BUF * *Packet*

)

Check whether this IP child accepts the packet.

Parameters:

[in]	*IpInstance*	The IP child to check.
[in]	*Head*	The IP header of the packet.
[in]	*Packet*	The data of the packet.

Return values:

TRUE	The child wants to receive the packet.
FALSE	The child does not want to receive the packet.

References IP6_CLIP_INFO::CastType, ICMP_V6_ERROR_MAX, IP6_GET_CLIP_INFO, Ip6IsExtsValid(), and Ip6Multicast.

Referenced by Ip6InstanceEnquePacket().

```
VOID Ip6InterfaceDeliverPacket ( IN IP6_SERVICE *        IpSb,
                                 IN IP6_INTERFACE *      IpIf
                               )
```

Deliver the packet for each IP6 child on the interface.

Parameters:

[in]	*IpSb*	The IP6 service instance that received the packet.
[in]	*IpIf*	The IP6 interface to deliver the packet.

References Ip6InstanceDeliverPacket().

Referenced by Ip6Demultiplex().

```
INTN Ip6InterfaceEnquePacket ( IN IP6_SERVICE *        IpSb,
                               IN EFI_IP6_HEADER *     Head,
                               IN NET_BUF *            Packet,
                               IN IP6_INTERFACE *      IpIf
                             )
```

Enqueue a received packet to all the IP children that share the same interface.

Parameters:

[in]	*IpSb*	The IP6 service instance that receive the packet.
[in]	*Head*	The header of the received packet.
[in]	*Packet*	The data of the received packet.
[in]	*IpIf*	The interface to enqueue the packet to.

Returns:

The number of the IP6 children that accepts the packet.

References IP6_CLIP_INFO::CastType, IP6_GET_CLIP_INFO, IP6_PROTOCOL_SIGNATURE, Ip6InstanceEnquePacket(), and Ip6Promiscuous.

Referenced by Ip6Demultiplex().

VOID EFIAPI Ip6IpSecFree (IN VOID * *Arg*)

The callback function for the net buffer that wraps the packet processed by IPsec. It releases the wrap packet and also signals IPsec to free the resources.

Parameters:

[in] *Arg* The wrap context.

References IP6_IPSEC_WRAP::IpSecRecycleSignal, and IP6_IPSEC_WRAP::Packet.

Referenced by Ip6IpSecProcessPacket().

```
EFI_STATUS Ip6IpSecProcessPacket ( IN IP6_SERVICE *              IpSb,
                                   IN OUT EFI_IP6_HEADER **      Head,
                                   IN OUT UINT8 *                LastHead,
                                   IN OUT NET_BUF **             Netbuf,
                                   IN OUT UINT8 **               ExtHdrs,
                                   IN OUT UINT32 *               ExtHdrsLen,
                                   IN EFI_IPSEC_TRAFFIC_DIR      Direction,
                                   IN VOID *                     Context
                                 )
```

The work function to locate the IPsec protocol to process the inbound or outbound IP packets. The process routine handles the packet with the following actions: bypass the packet, discard the packet, or protect the packet.

Parameters:

[in]	*IpSb*	The IP6 service instance.
[in,out]	*Head*	The caller-supplied IP6 header.
[in,out]	*LastHead*	The next header field of last IP header.
[in,out]	*Netbuf*	The IP6 packet to be processed by IPsec.
[in,out]	*ExtHdrs*	The caller-supplied options.
[in,out]	*ExtHdrsLen*	The length of the option.
[in]	*Direction*	The directionality in an SPD entry, EfiIPsecInBound, or EfiIPsecOutBound.
[in]	*Context*	The token's wrap.

Return values:

EFI_SUCCESS	The IPsec protocol is not available or disabled.
EFI_SUCCESS	The packet was bypassed, and all buffers remain the same.
EFI_SUCCESS	The packet was protected.
EFI_ACCESS_DENIED	The packet was discarded.
EFI_OUT_OF_RESOURCES	There are not suffcient resources to complete the operation.
EFI_BUFFER_TOO_SMALL	The number of non-empty blocks is bigger than the number of input data blocks when building a fragment table.

References IP6_GET_CLIP_INFO, IP6_MAX_HEADLEN, IP6_MAX_IPSEC_HEADLEN, Ip6FreeTxToken(), Ip6IpSecFree(), Ip6NtohHead(), IP6_IPSEC_WRAP::IpSecRecycleSignal, IP6_TXTOKEN_WRAP::IpSecRecycleSignal, mIpSec, mIpSec2Installed, IP6_IPSEC_WRAP::Packet, and IP6_TXTOKEN_WRAP::Packet.

Referenced by Ip6AcceptFrame(), and Ip6Output().

VOID EFIAPI Ip6OnFreeFragments (IN VOID * *Arg*)

Release all the fragments of the packet. This is the callback for the assembled packet's OnFree. It will free the assemble entry, which in turn frees all the fragments of the packet.

Parameters:

[in] *Arg* The assemble entry to free.

References Ip6FreeAssembleEntry().

Referenced by Ip6Reassemble().

VOID EFIAPI Ip6OnRecyclePacket (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

The signal handle of IP6's recycle event. It is called back when the upper layer releases the packet.

Parameters:

[in] *Event* The IP6's recycle event.

[in] *Context* The context of the handle, which is a IP6_RXDATA_WRAP.

References IP6_RXDATA_WRAP::IpInstance, IP6_RXDATA_WRAP::Link, IP6_RXDATA_WRAP::Packet, _IP6_PROTOCOL::RecycleLock, and IP6_RXDATA_WRAP::RxData.

Referenced by Ip6WrapRxData().

VOID Ip6PacketTimerTicking (IN IP6_SERVICE * *IpSb*)

Timeout the fragments, and the enqueued, and transmitted packets.

Parameters:

[in] *IpSb* The IP6 service instance to timeout.

References IP6_ASSEMBLE_ENTRY::Head, IP6_ASSEMLE_HASH_SIZE, IP6_GET_CLIP_INFO, Ip6FreeAssembleEntry(), Ip6SendIcmpError(), Ip6SentPacketTicking(), IP6_CLIP_INFO::Life, IP6_ASSEMBLE_ENTRY::Life, IP6_ASSEMBLE_ENTRY::Packet, _IP6_PROTOCOL::Received, and _IP6_PROTOCOL::TxTokens.

Referenced by Ip6TimerTicking().

EFI_STATUS Ip6PreProcessPacket (IN IP6_SERVICE * *IpSb*,
IN OUT NET_BUF ** *Packet*,
IN UINT32 *Flag*,
OUT UINT8 ** *Payload*,
OUT UINT8 ** *LastHead*,
OUT UINT32 * *ExtHdrsLen*,
OUT UINT32 * *UnFragmentLen*,
OUT BOOLEAN * *Fragmented*,

OUT EFI_IP6_HEADER ** *Head*
)

Pre-process the IPv6 packet. First validates the IPv6 packet, and then reassembles packet if it is necessary.

Parameters:

[in]	*IpSb*	The IP6 service instance.
[in,out]	*Packet*	The received IP6 packet to be processed.
[in]	*Flag*	The link layer flag for the packet received, such as multicast.
[out]	*Payload*	The pointer to the payload of the recieved packet. it starts from the first byte of the extension header.
[out]	*LastHead*	The pointer of NextHeader of the last extension header processed by IP6.
[out]	*ExtHdrsLen*	The length of the whole option.
[out]	*UnFragmentLen*	The length of unfragmented length of extension headers.
[out]	*Fragmented*	Indicate whether the packet is fragmented.
[out]	*Head*	The pointer to the EFI_IP6_Header.

Return values:

EFI_SUCCESS	The received packet is well format.
EFI_INVALID_PARAMETER	The received packet is malformed.

References IP6_CLIP_INFO::CastType, IP6_CLIP_INFO::End, IP6_CLIP_INFO::FormerNextHeader, _IP6_FRAGMENT_HEADER::FragmentOffset, IP6_CLIP_INFO::HeadLen, IP6_CLIP_INFO::Id, _IP6_FRAGMENT_HEADER::Identification, IP6_GET_CLIP_INFO, IP6_MIN_HEADLEN, Ip6FindMldEntry(), Ip6IsExtsValid(), Ip6IsOneOfSetAddress(), Ip6Multicast, Ip6NtohHead(), Ip6Promiscuous, Ip6Reassemble(), Ip6Unicast, IP6_CLIP_INFO::LastFrag, IP6_CLIP_INFO::Length, IP6_CLIP_INFO::LinkFlag, _IP6_FRAGMENT_HEADER::NextHeader, IP6_CLIP_INFO::NextHeader, IP6_CLIP_INFO::Start, and IP6_CLIP_INFO::Status.

Referenced by Ip6AcceptFrame().

NET_BUF* Ip6Reassemble (IN OUT IP6_ASSEMBLE_TABLE * *Table*,
IN NET_BUF * *Packet*
)

Reassemble the IP fragments. If all the fragments of the packet have been received, it will wrap the packet in a net buffer then return it to caller. If the packet can't be assembled, NULL is returned.

Parameters:

[in,out]	*Table*	The assemble table used. A new assemble entry will be created if the Packet is from a new chain of fragments.
[in]	*Packet*	The fragment to assemble. It might be freed if the fragment can't be re-assembled.

Returns:

NULL if the packet can't be reassembled. The pointer to the just assembled packet if all the fragments of the packet have arrived.

References IP6_ASSEMBLE_ENTRY::CurLen, IP6_ASSEMBLE_ENTRY::Dst, IP6_CLIP_INFO::End, IP6_CLIP_INFO::FormerNextHeader, IP6_ASSEMBLE_ENTRY::Fragments, IP6_ASSEMBLE_ENTRY::Head, IP6_CLIP_INFO::HeadLen, IP6_ASSEMBLE_ENTRY::Id, IP6_CLIP_INFO::Id, IP6_ASSEMBLE_ENTRY::Info, IP6_ASSEMBLE_HASH, IP6_GET_CLIP_INFO,

Ip6CreateAssembleEntry(), Ip6FreeAssembleEntry(), Ip6NtohHead(), Ip6OnFreeFragments(), Ip6TrimPacket(), IP6_CLIP_INFO::LastFrag, IP6_CLIP_INFO::Length, IP6_ASSEMBLE_ENTRY::Link, IP6_CLIP_INFO::NextHeader, IP6_ASSEMBLE_ENTRY::Packet, IP6_ASSEMBLE_ENTRY::Src, IP6_CLIP_INFO::Start, and IP6_ASSEMBLE_ENTRY::TotalLen.

Referenced by Ip6PreProcessPacket().

```
EFI_STATUS EFIAPI Ip6SentPacketTicking ( IN NET_MAP *        Map,
                                         IN NET_MAP_ITEM *   Item,
                                         IN VOID *           Context
                                       )
```

Decrease the life of the transmitted packets. If it is decreased to zero, cancel the packet. This function is called by Ip6packetTimerTicking that provides timeout for both the received-but-not-delivered and transmitted-but-not-recycle packets.

Parameters:

[in]	*Map*	The IP6 child's transmit map.
[in]	*Item*	Current transmitted packet.
[in]	*Context*	Not used.

Return values:

EFI_SUCCESS	Always returns EFI_SUCCESS.

References _IP6_PROTOCOL::Interface, Ip6CancelPacket(), IP6_TXTOKEN_WRAP::IpInstance, IP6_TXTOKEN_WRAP::Life, and IP6_TXTOKEN_WRAP::Packet.

Referenced by Ip6PacketTimerTicking().

```
VOID Ip6TrimPacket ( IN OUT NET_BUF *  Packet,
                     IN INTN           Start,
                     IN INTN           End
                   )
```

Trim the packet to fit in [Start, End), and update per the packet information.

Parameters:

[in,out]	*Packet*	Packet to trim.
[in]	*Start*	The sequence of the first byte to fit in.
[in]	*End*	One beyond the sequence of last byte to fit in.

References IP6_CLIP_INFO::End, IP6_GET_CLIP_INFO, IP6_CLIP_INFO::Length, and IP6_CLIP_INFO::Start.

Referenced by Ip6Reassemble().

```
IP6_RXDATA_WRAP* Ip6WrapRxData ( IN IP6_PROTOCOL *  IpInstance,
                                 IN NET_BUF *       Packet
                               )
```

Wrap the received packet to a IP6_RXDATA_WRAP, which will be delivered to the upper layer. Each IP6 child that accepts the packet will get a not-shared copy of the packet which is wrapped in the IP6_RXDATA_WRAP. The IP6_RXDATA_WRAP->RxData is passed to the upper layer. The upper layer will signal the recycle event in it when it is done with the packet.

Parameters:

[in]	*IpInstance*	The IP6 child to receive the packet.
[in]	*Packet*	The packet to deliver up.

Returns:

NULL if it failed to wrap the packet; otherwise, the wrapper.

References IP6_RXDATA_WRAP_SIZE, Ip6NtohHead(), Ip6OnRecyclePacket(), IP6_RXDATA_WRAP::IpInstance, IP6_RXDATA_WRAP::Link, IP6_RXDATA_WRAP::Packet, and IP6_RXDATA_WRAP::RxData.

Referenced by Ip6InstanceDeliverPacket().

NetworkPkg/Ip6Dxe/Ip6Input.h File Reference

Data Structures

struct IP6_CLIP_INFO
struct IP6_ASSEMBLE_ENTRY
struct IP6_ASSEMBLE_TABLE

Defines

"memItemRight" valign="bottom">IP6_MIN_HEADLEN 40

#define IP6_MAX_HEADLEN 120
#define IP6_MAX_IPSEC_HEADLEN 54
#define IP6_ASSEMLE_HASH_SIZE 127
#define IP6_FRAGMENT_LIFE 60
#define IP6_MAX_PACKET_SIZE 65535
#define IP6_GET_CLIP_INFO(Packet) ((IP6_CLIP_INFO *) ((Packet)->ProtoData))
#define IP6_ASSEMBLE_HASH(Dst, Src, Id) ((*((UINT32 *) (Dst)) + *((UINT32 *) (Src)) + (Id)) % IP6_ASSEMLE_HASH_SIZE)
#define IP6_RXDATA_WRAP_SIZE(NumFrag) (sizeof (IP6_RXDATA_WRAP) + sizeof (EFI_IP6_FRAGMENT_DATA) * ((NumFrag) - 1))

Functions

VOID Ip6AcceptFrame (IN NET_BUF *Packet, IN EFI_STATUS IoStatus, IN UINT32 Flag, IN VOID *Context)
EFI_STATUS Ip6InstanceDeliverPacket (IN IP6_PROTOCOL *IpInstance)
EFI_STATUS Ip6IpSecProcessPacket (IN IP6_SERVICE *IpSb, IN OUT EFI_IP6_HEADER **Head, IN OUT UINT8 *LastHead, IN OUT NET_BUF **Netbuf, IN OUT UINT8 **ExtHdrs, IN OUT UINT32 *ExtHdrsLen, IN EFI_IPSEC_TRAFFIC_DIR Direction, IN VOID *Context)
VOID Ip6CreateAssembleTable (IN OUT IP6_ASSEMBLE_TABLE *Table)
VOID Ip6CleanAssembleTable (IN OUT IP6_ASSEMBLE_TABLE *Table)
EFI_STATUS Ip6Demultiplex (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
VOID Ip6PacketTimerTicking (IN IP6_SERVICE *IpSb)

Detailed Description

IP6 internal functions and definitions to process the incoming packets.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"paramtype">Dst,
Src,
Id) ((*((UINT32 *) (Dst)) + *((UINT32 *) (Src)) + (Id)) % IP6_ASSEMLE_HASH_SIZE)

Referenced by Ip6Reassemble().

#define IP6_ASSEMLE_HASH_SIZE 127

Referenced by Ip6CleanAssembleTable(), Ip6CreateAssembleTable(), and Ip6PacketTimerTicking().

#define IP6_FRAGMENT_LIFE 60
Lift time in seconds.

Referenced by Ip6CreateAssembleEntry().

#define IP6_GET_CLIP_INFO (Packet) ((IP6_CLIP_INFO *) ((Packet)->ProtoData))

Referenced by Ip6AcceptFrame(), Ip6InstanceDeliverPacket(), Ip6InstanceEnquePacket(), Ip6InstanceFrameAcceptable(), Ip6InterfaceEnquePacket(), Ip6IpSecProcessPacket(), Ip6PacketTimerTicking(), Ip6PreProcessPacket(), Ip6ProcessIcmpError(), Ip6Reassemble(), and Ip6TrimPacket().

#define IP6_MAX_HEADLEN 120

Referenced by EfiIp6Transmit(), Ip6IcmpReplyEcho(), Ip6IpSecProcessPacket(), and Ip6OnFrameReceivedDpc().

#define IP6_MAX_IPSEC_HEADLEN 54
8(ESP header) + 16(max IV) + 16(max padding) + 2(ESP tail) + 12(max ICV) = 54

Referenced by Ip6IpSecProcessPacket().

#define IP6_MAX_PACKET_SIZE 65535

#define IP6_MIN_HEADLEN 40

Referenced by Ip6PreProcessPacket().

#define IP6_RXDATA_WRAP_SIZE (NumFrag) (sizeof (IP6_RXDATA_WRAP) + sizeof (EFI_IP6_FRAGMENT_DATA) * ((NumFrag) - 1))

Referenced by Ip6WrapRxData().

Function Documentation

```
VOID Ip6AcceptFrame ( IN NET_BUF *      Packet,
                      IN EFI_STATUS     IoStatus,
                      IN UINT32         Flag,
                      IN VOID *         Context
                    )
```

The IP6 input routine. It is called by the IP6_INTERFACE when an IP6 fragment is received from MNP.

Parameters:

[in]	*Packet*	The IP6 packet received.
[in]	*IoStatus*	The return status of receive request.
[in]	*Flag*	The link layer flag for the packet received, such as multicast.
[in]	*Context*	The IP6 service instance that own the MNP.

The IP6 input routine. It is called by the IP6_INTERFACE when an IP6 fragment is received from MNP.

Parameters:

[in]	*Packet*	The IP6 packet received.
[in]	*IoStatus*	The return status of receive request.
[in]	*Flag*	The link layer flag for the packet received, such as multicast.
[in]	*Context*	The IP6 service instance that owns the MNP.

References IP6_GET_CLIP_INFO, IP6_SERVICE_DESTROY, IP6_SERVICE_SIGNATURE, Ip6AcceptFrame(), Ip6Demultiplex(), Ip6IcmpHandle(), Ip6IpSecProcessPacket(), Ip6PreProcessPacket(), Ip6ReceiveFrame(), and _IP6_SERVICE::State.

Referenced by Ip6AcceptFrame(), and Ip6DriverBindingStart().

VOID Ip6CleanAssembleTable (IN OUT IP6_ASSEMBLE_TABLE * *Table*)

Clean up the assemble table: remove all the fragments and assemble entries.

Parameters:

`[in,out]` *Table* The assemble table to clean up.

Clean up the assemble table by removing all of the fragments and assemble entries.

Parameters:

`[in,out]` *Table* The assemble table to clean up.

References IP6_ASSEMLE_HASH_SIZE, and Ip6FreeAssembleEntry().

Referenced by Ip6CleanService().

VOID Ip6CreateAssembleTable (IN OUT IP6_ASSEMBLE_TABLE * *Table*)

Initialize an already allocated assemble table. This is generally the assemble table embedded in the IP6 service instance.

Parameters:

`[in,out]` *Table* The assemble table to initialize.

References IP6_ASSEMLE_HASH_SIZE.

Referenced by Ip6CreateService().

```
EFI_STATUS Ip6Demultiplex ( IN IP6_SERVICE *        IpSb,
                            IN EFI_IP6_HEADER *     Head,
                            IN NET_BUF *            Packet
                          )
```

Demultiple the packet. the packet delivery is processed in two passes. The first pass will enque a shared copy of the packet to each IP6 child that accepts the packet. The second pass will deliver a non-shared copy of the packet to each IP6 child that has pending receive requests. Data is copied if more than one child wants to consume the packet bacause each IP child need its own copy of the packet to make changes.

Parameters:

`[in]` *IpSb* The IP6 service instance that received the packet.
`[in]` *Head* The header of the received packet.
`[in]` *Packet* The data of the received packet.

Return values:

EFI_NOT_FOUND No IP child accepts the packet.
EFI_SUCCESS The packet is enqueued or delivered to some IP children.

De-multiplex the packet. the packet delivery is processed in two passes. The first pass will enqueue a shared copy of the packet to each IP6 child that accepts the packet. The second pass will deliver a non-shared copy of the packet to each IP6 child that has pending receive requests. Data is copied if more than one child wants to consume the packet, because each IP child needs its own copy of the packet to make changes.

Parameters:

`[in]` *IpSb* The IP6 service instance that received the packet.
`[in]` *Head* The header of the received packet.
`[in]` *Packet* The data of the received packet.

Return values:

EFI_NOT_FOUND No IP child accepts the packet.
EFI_SUCCESS The packet is enqueued or delivered to some IP children.

References _IP6_INTERFACE::Configured, Ip6InterfaceDeliverPacket(), and Ip6InterfaceEnquePacket().

Referenced by Ip6AcceptFrame(), Ip6ProcessIcmpError(), and Ip6ProcessIcmpInformation().

EFI_STATUS Ip6InstanceDeliverPacket (IN IP6_PROTOCOL * *IpInstance*)

Deliver the received packets to upper layer if there are both received requests and enqueued packets. If the enqueued packet is shared, it will duplicate it to a non-shared packet, release the shared packet, then deliver the non-shared packet up.

Parameters:

[in] *IpInstance* The IP child to deliver the packet up.

Return values:

EFI_OUT_OF_RESOURCES Failed to allocate resources to deliver the packets.
EFI_SUCCESS All the enqueued packets that can be delivered are delivered up.

Deliver the received packets to the upper layer if there are both received requests and enqueued packets. If the enqueued packet is shared, it will duplicate it to a non-shared packet, release the shared packet, then deliver the non-shared packet up.

Parameters:

[in] *IpInstance* The IP child to deliver the packet up.

Return values:

EFI_OUT_OF_RESOURCES Failed to allocate resources to deliver the packets.
EFI_SUCCESS All the enqueued packets that can be delivered are delivered up.

References IP6_GET_CLIP_INFO, Ip6WrapRxData(), IP6_RXDATA_WRAP::Link, and IP6_RXDATA_WRAP::RxData.

Referenced by EfiIp6Receive(), and Ip6InterfaceDeliverPacket().

```
EFI_STATUS Ip6IpSecProcessPacket ( IN IP6_SERVICE *                IpSb,
                                   IN OUT EFI_IP6_HEADER **        Head,
                                   IN OUT UINT8 *                  LastHead,
                                   IN OUT NET_BUF **               Netbuf,
                                   IN OUT UINT8 **                 ExtHdrs,
                                   IN OUT UINT32 *                 ExtHdrsLen,
                                   IN EFI_IPSEC_TRAFFIC_DIR        Direction,
                                   IN VOID *                       Context
                                 )
```

The work function to locate the IPsec protocol to process the inbound or outbound IP packets. The process routine handles the packet with the following actions: bypass the packet, discard the packet, or protect the packet.

Parameters:

[in] *IpSb* The IP6 service instance.

[in,out]	*Head*	The caller-supplied IP6 header.
[in,out]	*LastHead*	The next header field of last IP header.
[in,out]	*Netbuf*	The IP6 packet to be processed by IPsec.
[in,out]	*ExtHdrs*	The caller-supplied options.
[in,out]	*ExtHdrsLen*	The length of the option.
[in]	*Direction*	The directionality in an SPD entry, EfiIPsecInBound, or EfiIPsecOutBound.
[in]	*Context*	The token's wrap.

Return values:

EFI_SUCCESS	The IPsec protocol is not available or disabled.
EFI_SUCCESS	The packet was bypassed, and all buffers remain the same.
EFI_SUCCESS	The packet was protected.
EFI_ACCESS_DENIED	The packet was discarded.
EFI_OUT_OF_RESOURCES	There are not suffcient resources to complete the operation.
EFI_BUFFER_TOO_SMALL	The number of non-empty blocks is bigger than the number of input data blocks when building a fragment table.

References IP6_GET_CLIP_INFO, IP6_MAX_HEADLEN, IP6_MAX_IPSEC_HEADLEN, Ip6FreeTxToken(), Ip6IpSecFree(), Ip6NtohHead(), IP6_IPSEC_WRAP::IpSecRecycleSignal, IP6_TXTOKEN_WRAP::IpSecRecycleSignal, mIpSec, mIpSec2Installed, IP6_IPSEC_WRAP::Packet, and IP6_TXTOKEN_WRAP::Packet.

Referenced by Ip6AcceptFrame(), and Ip6Output().

VOID Ip6PacketTimerTicking (IN IP6_SERVICE * *IpSb*)

Timeout the fragmented, enqueued, and transmitted packets.

Parameters:

[in] *IpSb* The IP6 service instance to timeout.

Timeout the fragments, and the enqueued, and transmitted packets.

Parameters:

[in] *IpSb* The IP6 service instance to timeout.

References IP6_ASSEMBLE_ENTRY::Head, IP6_ASSEMLE_HASH_SIZE, IP6_GET_CLIP_INFO, Ip6FreeAssembleEntry(), Ip6SendIcmpError(), Ip6SentPacketTicking(), IP6_CLIP_INFO::Life, IP6_ASSEMBLE_ENTRY::Life, IP6_ASSEMBLE_ENTRY::Packet, _IP6_PROTOCOL::Received, and _IP6_PROTOCOL::TxTokens.

Referenced by Ip6TimerTicking().

- Globals

NetworkPkg/Ip6Dxe/Ip6Mld.c File Reference

Functions

IP6_MLD_GROUP * Ip6CreateMldEntry (IN OUT IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *MulticastAddr, IN UINT32 DelayTimer)
IP6_MLD_GROUP * Ip6FindMldEntry (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *MulticastAddr)
INTN Ip6FindMac (IN IP6_MLD_SERVICE_DATA *MldCtrl, IN EFI_MAC_ADDRESS *Mac)
EFI_STATUS Ip6SendMldReport (IN IP6_SERVICE *IpSb, IN IP6_INTERFACE *Interface, IN EFI_IPv6_ADDRESS *MulticastAddr)
EFI_STATUS Ip6SendMldDone (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *MulticastAddr)
EFI_STATUS Ip6InitMld (IN IP6_SERVICE *IpSb)
EFI_STATUS Ip6CombineGroups (IN OUT IP6_PROTOCOL *IpInstance, IN EFI_IPv6_ADDRESS *Group)
EFI_STATUS Ip6RemoveGroup (IN OUT IP6_PROTOCOL *IpInstance, IN EFI_IPv6_ADDRESS *Group)
EFI_STATUS Ip6JoinGroup (IN IP6_SERVICE *IpSb, IN IP6_INTERFACE *Interface, IN EFI_IPv6_ADDRESS *Address)
EFI_STATUS Ip6LeaveGroup (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *Address)
EFI_STATUS Ip6Groups (IN IP6_PROTOCOL *IpInstance, IN BOOLEAN JoinFlag, IN EFI_IPv6_ADDRESS *GroupAddress)
EFI_STATUS Ip6UpdateDelayTimer (IN IP6_SERVICE *IpSb, IN UINT16 MaxRespDelay, IN EFI_IPv6_ADDRESS *MulticastAddr, IN OUT IP6_MLD_GROUP *Group)
EFI_STATUS Ip6ProcessMldQuery (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
EFI_STATUS Ip6ProcessMldReport (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
VOID Ip6MldTimerTicking (IN IP6_SERVICE *IpSb)

Detailed Description

Multicast Listener Discovery support routines.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS Ip6CombineGroups (IN OUT IP6_PROTOCOL * *IpInstance*,

IN EFI_IPv6_ADDRESS * *Group*
)

Add a group address to the array of group addresses. The caller should make sure that no duplicated address existed in the array.

Parameters:

`[in,out]`	*IpInstance*	Points to an IP6_PROTOCOL instance.
`[in]`	*Group*	The IP6 multicast address to add.

Return values:

EFI_OUT_OF_RESOURCES	There are not sufficient resources to complete the operation.
EFI_SUCESS	The address is added to the group address array.

References IP6_PROTOCOL_SIGNATURE.

Referenced by Ip6Groups().

IP6_MLD_GROUP* Ip6CreateMldEntry (
IN OUT IP6_SERVICE * *IpSb*,
IN EFI_IPv6_ADDRESS * *MulticastAddr*,
IN UINT32 *DelayTimer*
)

Create a IP6_MLD_GROUP list entry node and record to IP6 service binding data.

Parameters:

`[in,out]`	*IpSb*	Points to IP6 service binding instance.
`[in]`	*MulticastAddr*	The IPv6 multicast address to be recorded.
`[in]`	*DelayTimer*	The maximum allowed delay before sending a responding report, in units of milliseconds.

Returns:

The created IP6_ML_GROUP list entry or NULL.

References IP6_MLD_GROUP::Address, IP6_MLD_GROUP::DelayTimer, IP6_SERVICE_SIGNATURE, IP6_MLD_GROUP::Link, IP6_MLD_GROUP::RefCnt, and IP6_MLD_GROUP::SendByUs.

Referenced by Ip6InitMld(), and Ip6JoinGroup().

INTN Ip6FindMac (IN IP6_MLD_SERVICE_DATA * *MldCtrl*,
IN EFI_MAC_ADDRESS * *Mac*
)

Count the number of IP6 multicast groups that are mapped to the same MAC address. Several IP6 multicast address may be mapped to the same MAC address.

Parameters:

`[in]`	*MldCtrl*	The MLD control block to search in.
`[in]`	*Mac*	The MAC address to search.

Returns:

The number of the IP6 multicast group that mapped to the same multicast group Mac.

References IP6_MLD_GROUP::Mac.

Referenced by Ip6LeaveGroup().

```
IP6_MLD_GROUP* Ip6FindMldEntry ( IN IP6_SERVICE *        IpSb,
                                 IN
                                 EFI_IPv6_ADDRESS *      MulticastAddr
                                 )
```

Search a IP6_MLD_GROUP list entry node from a list array.

Parameters:

[in]	*IpSb*	Points to IP6 service binding instance.
[in]	*MulticastAddr*	The IPv6 multicast address to be searched.

Returns:

The found IP6_ML_GROUP list entry or NULL.

References IP6_MLD_GROUP::Address, and IP6_SERVICE_SIGNATURE.

Referenced by Ip6JoinGroup(), Ip6LeaveGroup(), Ip6PreProcessPacket(), Ip6ProcessMldQuery(), and Ip6ProcessMldReport().

```
EFI_STATUS Ip6Groups ( IN IP6_PROTOCOL *   IpInstance,
                       IN BOOLEAN          JoinFlag,
                       IN
                       EFI_IPv6_ADDRESS    GroupAddress
                       *
                       )
```

Worker function for EfiIp6Groups(). The caller should make sure that the parameters are valid.

Parameters:

[in]	*IpInstance*	The IP6 child to change the setting.
[in]	*JoinFlag*	TRUE to join the group, otherwise leave it.
[in]	*GroupAddress*	The target group address. If NULL, leave all the group addresses.

Return values:

EFI_ALREADY_STARTED	Wants to join the group, but is already a member of it
EFI_OUT_OF_RESOURCES	Failed to allocate sufficient resources.
EFI_DEVICE_ERROR	Failed to set the group configuraton.
EFI_SUCCESS	Successfully updated the group setting.
EFI_NOT_FOUND	Try to leave the group which it isn't a member.

References Ip6CombineGroups(), Ip6JoinGroup(), Ip6LeaveGroup(), and Ip6RemoveGroup().

Referenced by EfiIp6Groups(), and Ip6CleanProtocol().

EFI_STATUS Ip6InitMld (IN IP6_SERVICE * *IpSb*)

Init the MLD data of the IP6 service instance. Configure MNP to receive ALL SYSTEM multicast.

Parameters:

[in] *IpSb* The IP6 service whose MLD is to be initialized.

Return values:

EFI_OUT_OF_RESOURCES There are not sufficient resourcet to complete the operation.
EFI_SUCCESS The MLD module successfully initialized.

References IP6_INFINIT_LIFETIME, IP6_LINK_LOCAL_SCOPE, Ip6CreateMldEntry(), Ip6GetMulticastMac(), Ip6SetToAllNodeMulticast(), IP6_MLD_GROUP::Link, and IP6_MLD_GROUP::Mac.

Referenced by Ip6CreateService().

EFI_STATUS Ip6JoinGroup (IN IP6_SERVICE * *IpSb*,
IN IP6_INTERFACE * *Interface*,
IN EFI_IPv6_ADDRESS * *Address*
)

Join the multicast group on behalf of this IP6 service binding instance.

Parameters:

[in] *IpSb* The IP6 service binding instance.
[in] *Interface* Points to an IP6_INTERFACE structure.
[in] *Address* The group address to join.

Return values:

EFI_SUCCESS Successfully join the multicast group.
EFI_OUT_OF_RESOURCES Failed to allocate resources.
Others Failed to join the multicast group.

References IP6_UNSOLICITED_REPORT_INTERVAL, Ip6CreateMldEntry(), Ip6FindMldEntry(), Ip6GetMulticastMac(), Ip6SendMldReport(), IP6_MLD_GROUP::Link, IP6_MLD_GROUP::Mac, IP6_MLD_GROUP::RefCnt, and IP6_MLD_GROUP::SendByUs.

Referenced by Ip6Groups(), and Ip6InitDADProcess().

EFI_STATUS Ip6LeaveGroup (IN IP6_SERVICE * *IpSb*,
IN EFI_IPv6_ADDRESS * *Address*
)

Leave the IP6 multicast group.

Parameters:

[in] *IpSb* The IP6 service binding instance.
[in] *Address* The group address to leave.

Return values:

EFI_NOT_FOUND The IP6 service instance isn't in the group.
EFI_SUCCESS Successfully leave the multicast group..

Others Failed to leave the multicast group.

References Ip6FindMac(), Ip6FindMldEntry(), Ip6SendMldDone(), IP6_MLD_GROUP::Link, IP6_MLD_GROUP::Mac, IP6_MLD_GROUP::RefCnt, and IP6_MLD_GROUP::SendByUs.

Referenced by Ip6CleanService(), Ip6Groups(), Ip6OnDADFinished(), and Ip6RemoveAddr().

VOID Ip6MldTimerTicking (IN IP6_SERVICE * *IpSb*)

The heartbeat timer of MLD module. It sends out a solicited MLD report when DelayTimer expires.

Parameters:

[in] *IpSb* The IP6 service binding instance.

References IP6_MLD_GROUP::Address, IP6_MLD_GROUP::DelayTimer, and Ip6SendMldReport().

Referenced by Ip6TimerTicking().

EFI_STATUS Ip6ProcessMldQuery (IN IP6_SERVICE * *IpSb*,
IN EFI_IP6_HEADER * *Head*,
IN NET_BUF * *Packet*
)

Process the Multicast Listener Query message.

Parameters:

[in] *IpSb* The IP service that received the packet.
[in] *Head* The IP head of the MLD query packet.
[in] *Packet* The content of the MLD query packet with IP head removed.

Return values:

EFI_SUCCESS The MLD query packet processed successfully.
EFI_INVALID_PARAMETER The packet is invalid.
Others Failed to process the packet.

References IP6_MLD_GROUP::Address, IP6_LINK_LOCAL_SCOPE, Ip6FindMldEntry(), Ip6SetToAllNodeMulticast(), and Ip6UpdateDelayTimer().

Referenced by Ip6ProcessIcmpInformation().

EFI_STATUS Ip6ProcessMldReport (IN IP6_SERVICE * *IpSb*,
IN EFI_IP6_HEADER * *Head*,
IN NET_BUF * *Packet*
)

Process the Multicast Listener Report message.

Parameters:

[in] *IpSb* The IP service that received the packet.
[in] *Head* The IP head of the MLD report packet.
[in] *Packet* The content of the MLD report packet with IP head removed.

Return values:

EFI_SUCCESS	The MLD report packet processed successfully.
EFI_INVALID_PARAMETER	The packet is invalid.

References IP6_MLD_GROUP::DelayTimer, Ip6FindMldEntry(), and IP6_MLD_GROUP::SendByUs.

Referenced by Ip6ProcessIcmpInformation().

```
EFI_STATUS Ip6RemoveGroup ( IN OUT IP6_PROTOCOL *   IpInstance,
                            IN EFI_IPv6_ADDRESS *   Group
                          )
```

Remove a group address from the array of group addresses. Although the function doesn't assume the byte order of Group, the network byte order is used by the caller.

Parameters:

`[in,out]`	*IpInstance*	Points to an IP6_PROTOCOL instance.
`[in]`	*Group*	The IP6 multicast address to remove.

Return values:

EFI_NOT_FOUND	Cannot find the to be removed group address.
EFI_SUCCESS	The group address was successfully removed.

Referenced by Ip6Groups().

```
EFI_STATUS Ip6SendMldDone ( IN IP6_SERVICE *        IpSb,
                            IN
                            EFI_IPv6_ADDRESS        MulticastAddr
                            *
                          )
```

Generate MLD Done message and send it out to MulticastAddr.

Parameters:

`[in]`	*IpSb*	The IP service to send the packet.
`[in]`	*MulticastAddr*	The specific IPv6 multicast address to which the message sender is ceasing to listen.

Return values:

EFI_OUT_OF_RESOURCES	There are not sufficient resources to complete the operation.
EFI_SUCCESS	The MLD report message was successfully sent out.

References IP6_MLD_HEAD::Group, IP6_MLD_HEAD::Head, IP6_LINK_LOCAL_SCOPE, IP6_SERVICE_SIGNATURE, Ip6FillHopByHop(), Ip6Output(), Ip6SetToAllNodeMulticast(), and Ip6SysPacketSent().

Referenced by Ip6LeaveGroup().

```
EFI_STATUS Ip6SendMldReport ( IN IP6_SERVICE *          IpSb,
                              IN IP6_INTERFACE *        Interface,
                              IN EFI_IPv6_ADDRESS *     MulticastAddr
```

)

Generate MLD report message and send it out to MulticastAddr.

Parameters:

[in]	*IpSb*	The IP service to send the packet.
[in]	*Interface*	The IP interface to send the packet. If NULL, a system interface will be selected.
[in]	*MulticastAddr*	The specific IPv6 multicast address to which the message sender is listening.

Return values:

EFI_OUT_OF_RESOURCES	There are not sufficient resources to complete the operation.
EFI_SUCCESS	The MLD report message was successfully sent out.

References IP6_MLD_HEAD::Group, IP6_MLD_HEAD::Head, IP6_SERVICE_SIGNATURE, Ip6FillHopByHop(), Ip6Output(), and Ip6SysPacketSent().

Referenced by Ip6JoinGroup(), Ip6MldTimerTicking(), and Ip6UpdateDelayTimer().

```
EFI_STATUS Ip6UpdateDelayTimer ( IN IP6_SERVICE *          IpSb,
                                 IN UINT16                 MaxRespDelay,
                                 IN EFI_IPv6_ADDRESS *     MulticastAddr,
                                 IN OUT IP6_MLD_GROUP *    Group
                               )
```

Set a random value of the delay timer for the multicast address from the range [0, Maximum Response Delay]. If a timer for any address is already running, it is reset to the new random value only if the requested Maximum Response Delay is less than the remaining value of the running timer. If the Query packet specifies a Maximum Response Delay of zero, each timer is effectively set to zero, and the action specified below for timer expiration is performed immediately.

Parameters:

[in]	*IpSb*	The IP6 service binding instance.
[in]	*MaxRespDelay*	The Maximum Response Delay, in milliseconds.
[in]	*MulticastAddr*	The multicast address.
[in,out]	*Group*	Points to a IP6_MLD_GROUP list entry node.

Return values:

EFI_SUCCESS	The delay timer is successfully updated or timer expiration is performed immediately.
Others	Failed to send out MLD report message.

References Ip6SendMldReport().

Referenced by Ip6ProcessMldQuery().

NetworkPkg/Ip6Dxe/Ip6Mld.h File Reference

Data Structures

struct IP6_MLD_HEAD
struct IP6_MLD_GROUP
struct IP6_MLD_SERVICE_DATA

Defines

"memItemRight" valign="bottom">IP6_UNSOLICITED_REPORT_INTERVAL 10

Functions

IP6_MLD_GROUP * Ip6FindMldEntry (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *MulticastAddr)
EFI_STATUS Ip6InitMld (IN IP6_SERVICE *IpSb)
EFI_STATUS Ip6JoinGroup (IN IP6_SERVICE *IpSb, IN IP6_INTERFACE *Interface, IN EFI_IPv6_ADDRESS *Address)
EFI_STATUS Ip6LeaveGroup (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *Address)
EFI_STATUS Ip6Groups (IN IP6_PROTOCOL *IpInstance, IN BOOLEAN JoinFlag, IN EFI_IPv6_ADDRESS *GroupAddress)
EFI_STATUS Ip6ProcessMldQuery (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
EFI_STATUS Ip6ProcessMldReport (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
VOID Ip6MldTimerTicking (IN IP6_SERVICE *IpSb)

Detailed Description

Multicast Listener Discovery support routines.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Ip6JoinGroup().

Function Documentation

```
IP6_MLD_GROUP* Ip6FindMldEntry ( IN IP6_SERVICE *        IpSb,
                                 IN EFI_IPv6_ADDRESS *   MulticastAddr
                               )
```

Search a IP6_MLD_GROUP list entry node from a list array.

Parameters:

[in]	*IpSb*	Points to an IP6 service binding instance.
[in]	*MulticastAddr*	The IPv6 multicast address to be searched.

Returns:

The found IP6_ML_GROUP list entry or NULL.

Search a IP6_MLD_GROUP list entry node from a list array.

Parameters:

[in]	*IpSb*	Points to IP6 service binding instance.
[in]	*MulticastAddr*	The IPv6 multicast address to be searched.

Returns:

The found IP6_ML_GROUP list entry or NULL.

References IP6_MLD_GROUP::Address, and IP6_SERVICE_SIGNATURE.

Referenced by Ip6JoinGroup(), Ip6LeaveGroup(), Ip6PreProcessPacket(), Ip6ProcessMldQuery(), and Ip6ProcessMldReport().

```
EFI_STATUS Ip6Groups ( IN IP6_PROTOCOL *  IpInstance,
                       IN BOOLEAN         JoinFlag,
                       IN                 GroupAddress
```

EFI_IPv6_ADDRESS
*
)

Worker function for EfiIp6Groups(). The caller should verify that the parameters are valid.

Parameters:

[in]	*IpInstance*	The IP6 child to change the setting.
[in]	*JoinFlag*	TRUE to join the group, otherwise leave it.
[in]	*GroupAddress*	The target group address. If NULL, leave all the group addresses.

Return values:

EFI_ALREADY_STARTED	Wants to join the group, but is already a member of it.
EFI_OUT_OF_RESOURCES	Failed to allocate some resources.
EFI_DEVICE_ERROR	Failed to set the group configuraton.
EFI_SUCCESS	Successfully updated the group setting.
EFI_NOT_FOUND	Tried to leave a group of whom it isn't a member.

Worker function for EfiIp6Groups(). The caller should make sure that the parameters are valid.

Parameters:

[in]	*IpInstance*	The IP6 child to change the setting.
[in]	*JoinFlag*	TRUE to join the group, otherwise leave it.
[in]	*GroupAddress*	The target group address. If NULL, leave all the group addresses.

Return values:

EFI_ALREADY_STARTED	Wants to join the group, but is already a member of it
EFI_OUT_OF_RESOURCES	Failed to allocate sufficient resources.
EFI_DEVICE_ERROR	Failed to set the group configuraton.
EFI_SUCCESS	Successfully updated the group setting.
EFI_NOT_FOUND	Try to leave the group which it isn't a member.

References Ip6CombineGroups(), Ip6JoinGroup(), Ip6LeaveGroup(), and Ip6RemoveGroup().

Referenced by EfiIp6Groups(), and Ip6CleanProtocol().

EFI_STATUS Ip6InitMld (IN IP6_SERVICE * *IpSb*)

Init the MLD data of the IP6 service instance, configure MNP to receive ALL SYSTEM multicasts.

Parameters:

[in]	*IpSb*	The IP6 service whose MLD is to be initialized.

Return values:

EFI_OUT_OF_RESOURCES	There are not sufficient resources to complete the operation.
EFI_SUCCESS	The MLD module successfully initialized.

Init the MLD data of the IP6 service instance. Configure MNP to receive ALL SYSTEM multicast.

Parameters:

[in]	*IpSb*	The IP6 service whose MLD is to be initialized.

Return values:

EFI_OUT_OF_RESOURCES	There are not sufficient resourcet to complete the operation.
EFI_SUCCESS	The MLD module successfully initialized.

References IP6_INFINIT_LIFETIME, IP6_LINK_LOCAL_SCOPE, Ip6CreateMldEntry(), Ip6GetMulticastMac(), Ip6SetToAllNodeMulticast(), IP6_MLD_GROUP::Link, and IP6_MLD_GROUP::Mac.

Referenced by Ip6CreateService().

```
EFI_STATUS Ip6JoinGroup ( IN IP6_SERVICE *          IpSb,
                          IN IP6_INTERFACE *        Interface,
                          IN EFI_IPv6_ADDRESS *     Address
                        )
```

Join the multicast group on behalf of this IP6 service binding instance.

Parameters:

[in]	*IpSb*	The IP6 service binding instance.
[in]	*Interface*	Points to an IP6_INTERFACE structure.
[in]	*Address*	The group address to join.

Return values:

EFI_SUCCESS	Successfully joined the multicast group.
EFI_OUT_OF_RESOURCES	Failed to allocate resources.
Others	Failed to join the multicast group.

Join the multicast group on behalf of this IP6 service binding instance.

Parameters:

[in]	*IpSb*	The IP6 service binding instance.
[in]	*Interface*	Points to an IP6_INTERFACE structure.
[in]	*Address*	The group address to join.

Return values:

EFI_SUCCESS	Successfully join the multicast group.
EFI_OUT_OF_RESOURCES	Failed to allocate resources.
Others	Failed to join the multicast group.

References IP6_UNSOLICITED_REPORT_INTERVAL, Ip6CreateMldEntry(), Ip6FindMldEntry(), Ip6GetMulticastMac(), Ip6SendMldReport(), IP6_MLD_GROUP::Link, IP6_MLD_GROUP::Mac, IP6_MLD_GROUP::RefCnt, and IP6_MLD_GROUP::SendByUs.

Referenced by Ip6Groups(), and Ip6InitDADProcess().

```
EFI_STATUS Ip6LeaveGroup ( IN IP6_SERVICE *          IpSb,
                           IN EFI_IPv6_ADDRESS *     Address
                         )
```

Leave the IP6 multicast group.

Parameters:

[in]	*IpSb*	The IP6 service binding instance.

[in] *Address* The group address to leave.

Return values:

EFI_NOT_FOUND The IP6 service instance isn't in the group.
EFI_SUCCESS Successfully left the multicast group.
Others Failed to leave the multicast group.

Leave the IP6 multicast group.

Parameters:

[in] *IpSb* The IP6 service binding instance.
[in] *Address* The group address to leave.

Return values:

EFI_NOT_FOUND The IP6 service instance isn't in the group.
EFI_SUCCESS Successfully leave the multicast group..
Others Failed to leave the multicast group.

References Ip6FindMac(), Ip6FindMldEntry(), Ip6SendMldDone(), IP6_MLD_GROUP::Link, IP6_MLD_GROUP::Mac, IP6_MLD_GROUP::RefCnt, and IP6_MLD_GROUP::SendByUs.

Referenced by Ip6CleanService(), Ip6Groups(), Ip6OnDADFinished(), and Ip6RemoveAddr().

VOID Ip6MldTimerTicking (IN IP6_SERVICE * *IpSb*)

The heartbeat timer of the MLD module. It sends out solicited MLD report when DelayTimer expires.

Parameters:

[in] *IpSb* The IP6 service binding instance.

The heartbeat timer of MLD module. It sends out a solicited MLD report when DelayTimer expires.

Parameters:

[in] *IpSb* The IP6 service binding instance.

References IP6_MLD_GROUP::Address, IP6_MLD_GROUP::DelayTimer, and Ip6SendMldReport().

Referenced by Ip6TimerTicking().

EFI_STATUS Ip6ProcessMldQuery (IN IP6_SERVICE * *IpSb*,
IN EFI_IP6_HEADER * *Head*,
IN NET_BUF * *Packet*
)

Process the Multicast Listener Query message.

Parameters:

[in] *IpSb* The IP service that received the packet.
[in] *Head* The IP head of the MLD query packet.
[in] *Packet* The content of the MLD query packet with IP head removed.

Return values:

EFI_SUCCESS The MLD query packet processed successfully.

EFI_INVALID_PARAMETER	The packet is invalid.
Others	Failed to process the packet.

References IP6_MLD_GROUP::Address, IP6_LINK_LOCAL_SCOPE, Ip6FindMldEntry(), Ip6SetToAllNodeMulticast(), and Ip6UpdateDelayTimer().

Referenced by Ip6ProcessIcmpInformation().

```
EFI_STATUS Ip6ProcessMldReport ( IN IP6_SERVICE *        IpSb,
                                 IN EFI_IP6_HEADER *     Head,
                                 IN NET_BUF *            Packet
                               )
```

Process the Multicast Listener Report message.

Parameters:

[in]	*IpSb*	The IP service that received the packet.
[in]	*Head*	The IP head of the MLD report packet.
[in]	*Packet*	The content of the MLD report packet with IP head removed.

Return values:

EFI_SUCCESS	The MLD report packet processed successfully.
EFI_INVALID_PARAMETER	The packet is invalid.

References IP6_MLD_GROUP::DelayTimer, Ip6FindMldEntry(), and IP6_MLD_GROUP::SendByUs.

Referenced by Ip6ProcessIcmpInformation().

NetworkPkg/Ip6Dxe/Ip6Nd.c File Reference

Functions

VOID Ip6UpdateReachableTime (IN OUT IP6_SERVICE *IpSb)

EFI_STATUS Ip6BuildEfiNeighborCache (IN IP6_PROTOCOL *IpInstance, OUT UINT32 *NeighborCount, OUT EFI_IP6_NEIGHBOR_CACHE **NeighborCache)

EFI_STATUS Ip6BuildPrefixTable (IN IP6_PROTOCOL *IpInstance, OUT UINT32 *PrefixCount, OUT EFI_IP6_ADDRESS_INFO **PrefixTable)

IP6_PREFIX_LIST_ENTRY * Ip6CreatePrefixListEntry (IN IP6_SERVICE *IpSb, IN BOOLEAN OnLinkOrAuto, IN UINT32 ValidLifetime, IN UINT32 PreferredLifetime, IN UINT8 PrefixLength, IN EFI_IPv6_ADDRESS *Prefix)

VOID Ip6DestroyPrefixListEntry (IN IP6_SERVICE *IpSb, IN IP6_PREFIX_LIST_ENTRY *PrefixEntry, IN BOOLEAN OnLinkOrAuto, IN BOOLEAN ImmediateDelete)

IP6_PREFIX_LIST_ENTRY * Ip6FindPrefixListEntry (IN IP6_SERVICE *IpSb, IN BOOLEAN OnLinkOrAuto, IN UINT8 PrefixLength, IN EFI_IPv6_ADDRESS *Prefix)

VOID Ip6CleanPrefixListTable (IN IP6_SERVICE *IpSb, IN LIST_ENTRY *ListHead)

VOID Ip6OnArpResolved (IN VOID *Context)

IP6_NEIGHBOR_ENTRY * Ip6CreateNeighborEntry (IN IP6_SERVICE *IpSb, IN IP6_ARP_CALLBACK CallBack, IN EFI_IPv6_ADDRESS *Ip6Address, IN EFI_MAC_ADDRESS *LinkAddress)

IP6_NEIGHBOR_ENTRY * Ip6FindNeighborEntry (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *Ip6Address)

EFI_STATUS Ip6FreeNeighborEntry (IN IP6_SERVICE *IpSb, IN IP6_NEIGHBOR_ENTRY *NeighborCache, IN BOOLEAN SendIcmpError, IN BOOLEAN FullFree, IN EFI_STATUS IoStatus, IN IP6_FRAME_TO_CANCEL FrameToCancel, IN VOID *Context)

IP6_DEFAULT_ROUTER * Ip6CreateDefaultRouter (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *Ip6Address, IN UINT16 RouterLifetime)

VOID Ip6DestroyDefaultRouter (IN IP6_SERVICE *IpSb, IN IP6_DEFAULT_ROUTER *DefaultRouter)

VOID Ip6CleanDefaultRouterList (IN IP6_SERVICE *IpSb)

IP6_DEFAULT_ROUTER * Ip6FindDefaultRouter (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *Ip6Address)

VOID Ip6OnDADFinished (IN BOOLEAN IsDadPassed, IN IP6_INTERFACE *IpIf, IN IP6_DAD_ENTRY *DadEntry)

EFI_STATUS Ip6InitDADProcess (IN IP6_INTERFACE *IpIf, IN IP6_ADDRESS_INFO *AddressInfo, IN IP6_DAD_CALLBACK Callback, IN VOID *Context)

IP6_DAD_ENTRY * Ip6FindDADEntry (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *Target, OUT IP6_INTERFACE **Interface)

EFI_STATUS Ip6SendRouterSolicit (IN IP6_SERVICE *IpSb, IN IP6_INTERFACE *Interface, IN EFI_IPv6_ADDRESS *SourceAddress, IN EFI_IPv6_ADDRESS *DestinationAddress, IN EFI_MAC_ADDRESS *SourceLinkAddress)

EFI_STATUS Ip6SendNeighborAdvertise (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *SourceAddress, IN EFI_IPv6_ADDRESS *DestinationAddress, IN EFI_IPv6_ADDRESS *TargetIp6Address, IN EFI_MAC_ADDRESS *TargetLinkAddress, IN BOOLEAN IsRouter, IN BOOLEAN Override, IN BOOLEAN Solicited)

EFI_STATUS

Ip6SendNeighborSolicit (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *SourceAddress, IN EFI_IPv6_ADDRESS *DestinationAddress, IN EFI_IPv6_ADDRESS *TargetIp6Address, IN EFI_MAC_ADDRESS *SourceLinkAddress)
EFI_STATUS Ip6ProcessNeighborSolicit (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
EFI_STATUS Ip6ProcessNeighborAdvertise (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
EFI_STATUS Ip6ProcessRouterAdvertise (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
EFI_STATUS Ip6ProcessRedirect (IN IP6_SERVICE *IpSb, IN EFI_IP6_HEADER *Head, IN NET_BUF *Packet)
EFI_STATUS Ip6AddNeighbor (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *TargetIp6Address, IN EFI_MAC_ADDRESS *TargetLinkAddress, IN UINT32 Timeout, IN BOOLEAN Override)
EFI_STATUS Ip6DelNeighbor (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *TargetIp6Address, IN EFI_MAC_ADDRESS *TargetLinkAddress, IN UINT32 Timeout, IN BOOLEAN Override)
VOID EFIAPI Ip6NdFasterTimerTicking (IN EFI_EVENT Event, IN VOID *Context)
VOID Ip6NdTimerTicking (IN IP6_SERVICE *IpSb)

Variables

EFI_MAC_ADDRESS mZeroMacAddress

Detailed Description

Implementation of Neighbor Discovery support routines.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS Ip6AddNeighbor ( IN IP6_SERVICE *          IpSb,
                            IN EFI_IPv6_ADDRESS *     TargetIp6Address,
                            IN EFI_MAC_ADDRESS *      TargetLinkAddress,
                            IN UINT32                 Timeout,
                            IN BOOLEAN                Override
                          )
```

Add Neighbor cache entries. It is a work function for EfiIp6Neighbors().

Parameters:

[in]	*IpSb*	The IP6 service binding instance.
[in]	*TargetIp6Address*	Pointer to Target IPv6 address.
[in]	*TargetLinkAddress*	Pointer to link-layer address of the target. Ignored if NULL.
[in]	*Timeout*	Time in 100-ns units that this entry will remain in the neighbor cache. It will be deleted after Timeout. A value of zero means that the entry is permanent. A non-zero value means that the entry is dynamic.
[in]	*Override*	If TRUE, the cached link-layer address of the matching entry will be overridden and updated; if FALSE, and if a corresponding cache entry already existed, EFI_ACCESS_DENIED will be returned.

Return values:

EFI_SUCCESS	The neighbor cache entry has been added.
EFI_OUT_OF_RESOURCES	Could not add the entry to the neighbor cache due to insufficient resources.
EFI_NOT_FOUND	TargetLinkAddress is NULL.
EFI_ACCESS_DENIED	The to-be-added entry is already defined in the neighbor cache, and that entry is tagged as un-overridden (when DeleteFlag is FALSE).

References _IP6_NEIGHBOR_ENTRY::Dynamic, IP6_GET_TICKS, IP6_INFINIT_LIFETIME, Ip6CreateNeighborEntry(), Ip6FindNeighborEntry(), Ip6OnArpResolved(), _IP6_NEIGHBOR_ENTRY::LinkAddress, _IP6_NEIGHBOR_ENTRY::State, and _IP6_NEIGHBOR_ENTRY::Ticks.

Referenced by EfiIp6Neighbors().

```
EFI_STATUS Ip6BuildEfiNeighborCache ( IN IP6_PROTOCOL *                IpInstance,
                                      OUT UINT32 *                     NeighborCount,
                                      OUT EFI_IP6_NEIGHBOR_CACHE **    NeighborCache
                                    )
```

Build a array of EFI_IP6_NEIGHBOR_CACHE to be returned to the caller. The number of EFI_IP6_NEIGHBOR_CACHE is also returned.

Parameters:

[in]	*IpInstance*	The pointer to IP6_PROTOCOL instance.
[out]	*NeighborCount*	The number of returned neighbor cache entries.
[out]	*NeighborCache*	The pointer to the array of EFI_IP6_NEIGHBOR_CACHE.

Return values:

EFI_SUCCESS	The EFI_IP6_NEIGHBOR_CACHE successfully built.
EFI_OUT_OF_RESOURCES	Failed to allocate the memory for the route table.

References IP6_PROTOCOL_SIGNATURE, _IP6_NEIGHBOR_ENTRY::LinkAddress, _IP6_NEIGHBOR_ENTRY::Neighbor, _IP6_SERVICE::NeighborTable, and _IP6_NEIGHBOR_ENTRY::State.

Referenced by EfiIp6GetModeData().

```
EFI_STATUS Ip6BuildPrefixTable ( IN IP6_PROTOCOL *                IpInstance,
                                 OUT UINT32 *                     PrefixCount,
                                 OUT EFI_IP6_ADDRESS_INFO **      PrefixTable
```

)

Build a array of EFI_IP6_ADDRESS_INFO to be returned to the caller. The number of prefix entries is also returned.

Parameters:

[in]	*IpInstance*	The pointer to IP6_PROTOCOL instance.
[out]	*PrefixCount*	The number of returned prefix entries.
[out]	*PrefixTable*	The pointer to the array of PrefixTable.

Return values:

EFI_SUCCESS	The prefix table successfully built.
EFI_OUT_OF_RESOURCES	Failed to allocate the memory for the prefix table.

References IP6_PROTOCOL_SIGNATURE, _IP6_SERVICE::OnlinkPrefix, _IP6_PREFIX_LIST_ENTRY::Prefix, and _IP6_PREFIX_LIST_ENTRY::PrefixLength.

Referenced by EfiIp6GetModeData().

VOID Ip6CleanDefaultRouterList (IN IP6_SERVICE * *IpSb*)

Clean an IP6 default router list.

Parameters:

[in]	*IpSb*	The pointer to the IP6_SERVICE instance.

References Ip6DestroyDefaultRouter().

Referenced by Ip6CleanService(), and Ip6ConfigOnPolicyChanged().

VOID Ip6CleanPrefixListTable (IN IP6_SERVICE * *IpSb*,
IN LIST_ENTRY * *ListHead*
)

Release the resource in the prefix list table, and destroy the list entry and corresponding addresses or route entries.

Parameters:

[in]	*IpSb*	The pointer to the IP6_SERVICE instance.
[in]	*ListHead*	The list entry head of the prefix list table.

References Ip6DestroyPrefixListEntry().

Referenced by Ip6CleanService(), and Ip6ConfigOnPolicyChanged().

IP6_DEFAULT_ROUTER* Ip6CreateDefaultRouter (IN IP6_SERVICE * *IpSb*,
IN EFI_IPv6_ADDRESS * *Ip6Address*,
IN UINT16 *RouterLifetime*
)

Allocate and initialize an IP6 default router entry.

Parameters:

[in]	*IpSb*	The pointer to the IP6_SERVICE instance.
[in]	*Ip6Address*	The IPv6 address of the default router.
[in]	*RouterLifetime*	The lifetime associated with the default router, in units of seconds.

Returns:

NULL if it failed to allocate memory for the default router node. Otherwise, point to the created default router node.

References IP6_SERVICE_SIGNATURE, Ip6CreateRouteEntry(), Ip6FindNeighborEntry(), _IP6_DEFAULT_ROUTER::Lifetime, _IP6_DEFAULT_ROUTER::Link, IP6_ROUTE_ENTRY::Link, _IP6_DEFAULT_ROUTER::NeighborCache, _IP6_DEFAULT_ROUTER::RefCnt, and _IP6_DEFAULT_ROUTER::Router.

Referenced by Ip6ConfigSetGateway(), and Ip6ProcessRouterAdvertise().

```
IP6_NEIGHBOR_ENTRY* Ip6CreateNeighborEntry ( IN IP6_SERVICE *          IpSb,
                                             IN IP6_ARP_CALLBACK       CallBack,
                                             IN EFI_IPv6_ADDRESS *     Ip6Address,
                                             IN EFI_MAC_ADDRESS *      LinkAddress
                                           )
```

Allocate and initialize an IP6 neighbor cache entry.

Parameters:

[in]	*IpSb*	The pointer to the IP6_SERVICE instance.
[in]	*CallBack*	The callback function to be called when address resolution is finished.
[in]	*Ip6Address*	Points to the IPv6 address of the neighbor.
[in]	*LinkAddress*	Points to the MAC address of the neighbor. Ignored if NULL.

Returns:

NULL if failed to allocate memory for the neighbor cache entry. Otherwise, point to the created neighbor cache entry.

References _IP6_NEIGHBOR_ENTRY::ArpFree, _IP6_NEIGHBOR_ENTRY::CallBack, _IP6_NEIGHBOR_ENTRY::Dynamic, _IP6_NEIGHBOR_ENTRY::Frames, _IP6_NEIGHBOR_ENTRY::Interface, IP6_MAX_MULTICAST_SOLICIT, IP6_SERVICE_SIGNATURE, Ip6FindDefaultRouter(), _IP6_NEIGHBOR_ENTRY::IsRouter, _IP6_NEIGHBOR_ENTRY::Link, _IP6_NEIGHBOR_ENTRY::LinkAddress, mZeroMacAddress, _IP6_NEIGHBOR_ENTRY::Neighbor, _IP6_DEFAULT_ROUTER::NeighborCache, _IP6_NEIGHBOR_ENTRY::RefCnt, _IP6_NEIGHBOR_ENTRY::State, and _IP6_NEIGHBOR_ENTRY::Transmit.

Referenced by Ip6AddNeighbor(), Ip6Output(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), Ip6ProcessRouterAdvertise(), and Ip6SendNeighborSolicit().

```
IP6_PREFIX_LIST_ENTRY* Ip6CreatePrefixListEntry ( IN IP6_SERVICE *          IpSb,
                                                  IN BOOLEAN                OnLinkOrAuto,
                                                  IN UINT32                 ValidLifetime,
                                                  IN UINT32                 PreferredLifetime,
                                                  IN UINT8                  PrefixLength,
                                                  IN EFI_IPv6_ADDRESS *     Prefix
```

)

Allocate and initialize a IP6 prefix list entry.

Parameters:

[in]	*IpSb*	The pointer to IP6_SERVICE instance.
[in]	*OnLinkOrAuto*	If TRUE, the entry is created for the on link prefix list. Otherwise, it is created for the autoconfiguration prefix list.
[in]	*ValidLifetime*	The length of time in seconds that the prefix is valid for the purpose of on-link determination.
[in]	*PreferredLifetime*	The length of time in seconds that addresses generated from the prefix via stateless address autoconfiguration remain preferred.
[in]	*PrefixLength*	The prefix length of the Prefix.
[in]	*Prefix*	The prefix address.

Returns:

NULL if it failed to allocate memory for the prefix node. Otherwise, point to the created or existing prefix list entry.

References IP6_ROUTE_ENTRY::Flag, IP6_DIRECT_ROUTE, IP6_SERVICE_SIGNATURE, Ip6CreateRouteEntry(), Ip6FindPrefixListEntry(), _IP6_PREFIX_LIST_ENTRY::Link, IP6_ROUTE_ENTRY::Link, _IP6_PREFIX_LIST_ENTRY::PreferredLifetime, _IP6_PREFIX_LIST_ENTRY::Prefix, _IP6_PREFIX_LIST_ENTRY::PrefixLength, _IP6_PREFIX_LIST_ENTRY::RefCnt, and _IP6_PREFIX_LIST_ENTRY::ValidLifetime.

Referenced by Ip6ConfigOnPolicyChanged(), Ip6ConfigSetMaunualAddress(), Ip6OnDADFinished(), and Ip6ProcessRouterAdvertise().

```
EFI_STATUS Ip6DelNeighbor ( IN IP6_SERVICE *          IpSb,
                            IN EFI_IPv6_ADDRESS *     TargetIp6Address,
                            IN EFI_MAC_ADDRESS *      TargetLinkAddress,
                            IN UINT32                 Timeout,
                            IN BOOLEAN                Override
                          )
```

Delete or update Neighbor cache entries. It is a work function for EfiIp6Neighbors().

Parameters:

[in]	*IpSb*	The IP6 service binding instance.
[in]	*TargetIp6Address*	Pointer to Target IPv6 address.
[in]	*TargetLinkAddress*	Pointer to link-layer address of the target. Ignored if NULL.
[in]	*Timeout*	Time in 100-ns units that this entry will remain in the neighbor cache. It will be deleted after Timeout. A value of zero means that the entry is permanent. A non-zero value means that the entry is dynamic.
[in]	*Override*	If TRUE, the cached link-layer address of the matching entry will be overridden and updated; if FALSE, and if a corresponding cache entry already existed, EFI_ACCESS_DENIED will be returned.

Return values:

EFI_SUCCESS	The neighbor cache entry has been updated or deleted.
EFI_NOT_FOUND	This entry is not in the neighbor cache.

References Ip6FindNeighborEntry(), and _IP6_NEIGHBOR_ENTRY::Link.

Referenced by EfiIp6Neighbors().

```
VOID Ip6DestroyDefaultRouter ( IN IP6_SERVICE *              IpSb,
                               IN IP6_DEFAULT_ROUTER *       DefaultRouter
                             )
```

Destroy an IP6 default router entry.

Parameters:

[in]	*IpSb*	The pointer to the IP6_SERVICE instance.
[in]	*DefaultRouter*	The to be destroyed IP6_DEFAULT_ROUTER.

References Ip6DelRoute().

Referenced by Ip6CleanDefaultRouterList(), Ip6ConfigSetGateway(), Ip6FreeNeighborEntry(), Ip6NdTimerTicking(), Ip6ProcessNeighborAdvertise(), and Ip6ProcessRouterAdvertise().

```
VOID Ip6DestroyPrefixListEntry ( IN IP6_SERVICE *                IpSb,
                                 IN IP6_PREFIX_LIST_ENTRY *      PrefixEntry,
                                 IN BOOLEAN                      OnLinkOrAuto,
                                 IN BOOLEAN                      ImmediateDelete
                               )
```

Destroy a IP6 prefix list entry.

Parameters:

[in]	*IpSb*	The pointer to IP6_SERVICE instance.
[in]	*PrefixEntry*	The to be destroyed prefix list entry.
[in]	*OnLinkOrAuto*	If TRUE, the entry is removed from on link prefix list. Otherwise remove from autoconfiguration prefix list.
[in]	*ImmediateDelete*	If TRUE, remove the entry directly. Otherwise, check the reference count to see whether it should be removed.

References _IP6_INTERFACE::AddressCount, _IP6_INTERFACE::AddressList, IP6_INTERFACE_SIGNATURE, Ip6DelRoute(), and Ip6RemoveAddr().

Referenced by Ip6CleanPrefixListTable(), Ip6ConfigSetMaunualAddress(), Ip6NdTimerTicking(), and Ip6ProcessRouterAdvertise().

```
IP6_DAD_ENTRY* Ip6FindDADEntry ( IN IP6_SERVICE *          IpSb,
                                 IN EFI_IPv6_ADDRESS *     Target,
                                 OUT IP6_INTERFACE **      Interface
                               )
```

Search IP6_DAD_ENTRY from the Duplicate Address Detection List.

Parameters:

[in]	*IpSb*	The pointer to the IP6_SERVICE instance.
[in]	*Target*	The address information which needs DAD performed .
[out]	*Interface*	If not NULL, output the IP6 interface that configures the tentative address.

Returns:

NULL if failed to find the matching DAD entry. Otherwise, point to the found DAD entry.

References _IP6_ADDRESS_INFO::Address, _IP6_DAD_ENTRY::AddressInfo, _IP6_INTERFACE::DupAddrDetectList, and IP6_DAD_ENTRY_SIGNATURE.

Referenced by Ip6InitDADProcess(), Ip6ProcessNeighborAdvertise(), Ip6ProcessNeighborSolicit(), and Ip6ProcessRouterAdvertise().

```
IP6_DEFAULT_ROUTER* Ip6FindDefaultRouter ( IN IP6_SERVICE *          IpSb,
                                           IN EFI_IPv6_ADDRESS *     Ip6Address
                                         )
```

Search a default router node from an IP6 default router list.

Parameters:

[in]	*IpSb*	The pointer to the IP6_SERVICE instance.
[in]	*Ip6Address*	The IPv6 address of the to be searched default router node.

Returns:

NULL if it failed to find the matching default router node. Otherwise, point to the found default router node.

References IP6_SERVICE_SIGNATURE, and _IP6_DEFAULT_ROUTER::Router.

Referenced by Ip6ConfigSetGateway(), Ip6CreateNeighborEntry(), Ip6FreeNeighborEntry(), Ip6ProcessNeighborAdvertise(), and Ip6ProcessRouterAdvertise().

```
IP6_NEIGHBOR_ENTRY* Ip6FindNeighborEntry ( IN IP6_SERVICE *      IpSb,
                                           IN
                                           EFI_IPv6_ADDRESS      Ip6Address
                                           *
                                         )
```

Search a IP6 neighbor cache entry.

Parameters:

[in]	*IpSb*	The pointer to the IP6_SERVICE instance.
[in]	*Ip6Address*	Points to the IPv6 address of the neighbor.

Returns:

NULL if it failed to find the matching neighbor cache entry. Otherwise, point to the found neighbor cache entry.

References IP6_SERVICE_SIGNATURE, and _IP6_NEIGHBOR_ENTRY::Neighbor.

Referenced by Ip6AddNeighbor(), Ip6CreateDefaultRouter(), Ip6DelNeighbor(), Ip6Output(), Ip6ProcessNeighborAdvertise(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), Ip6ProcessRouterAdvertise(), Ip6SendFrame(), and Ip6SendNeighborSolicit().

```
IP6_PREFIX_LIST_ENTRY* Ip6FindPrefixListEntry ( IN IP6_SERVICE *      IpSb,
                                                IN BOOLEAN            OnLinkOrAuto,
```

```
                                    IN UINT8                 PrefixLength,
                                    IN EFI_IPv6_ADDRESS *    Prefix
                                  )
```

Search the list array to find an IP6 prefix list entry.

Parameters:

[in]	*IpSb*	The pointer to IP6_SERVICE instance.
[in]	*OnLinkOrAuto*	If TRUE, the search the link prefix list, Otherwise search the autoconfiguration prefix list.
[in]	*PrefixLength*	The prefix length of the Prefix
[in]	*Prefix*	The prefix address.

Returns:

NULL if cannot find the IP6 prefix list entry. Otherwise, return the pointer to the IP6 prefix list entry.

References IP6_SERVICE_SIGNATURE, _IP6_PREFIX_LIST_ENTRY::Prefix, and _IP6_PREFIX_LIST_ENTRY::PrefixLength.

Referenced by Ip6ConfigSetMaunualAddress(), Ip6CreatePrefixListEntry(), Ip6IsAnycast(), and Ip6ProcessRouterAdvertise().

```
EFI_STATUS Ip6FreeNeighborEntry ( IN IP6_SERVICE *              IpSb,
                                  IN IP6_NEIGHBOR_ENTRY *       NeighborCache,
                                  IN BOOLEAN                    SendIcmpError,
                                  IN BOOLEAN                    FullFree,
                                  IN EFI_STATUS                 IoStatus,
                                  IN IP6_FRAME_TO_CANCEL        FrameToCancel,
                                  IN VOID *                     Context
                                )
```

Free a IP6 neighbor cache entry and remove all the frames on the address resolution queue that pass the FrameToCancel. That is, either FrameToCancel is NULL, or it returns true for the frame.

Parameters:

[in]	*IpSb*	The pointer to the IP6_SERVICE instance.
[in]	*NeighborCache*	The to be free neighbor cache entry.
[in]	*SendIcmpError*	If TRUE, send out ICMP error.
[in]	*FullFree*	If TRUE, remove the neighbor cache entry. Otherwise remove the pending frames.
[in]	*IoStatus*	The status returned to the cancelled frames' callback function.
[in]	*FrameToCancel*	Function to select which frame to cancel. This is an optional parameter that may be NULL.
[in]	*Context*	Opaque parameter to the FrameToCancel. Ignored if FrameToCancel is NULL.

Return values:

EFI_INVALID_PARAMETER	The input parameter is invalid.
EFI_SUCCESS	The operation finished successfully.

References IP6_LINK_TX_TOKEN::CallBack, IP6_LINK_TX_TOKEN::Context, Ip6DestroyDefaultRouter(), Ip6FindDefaultRouter(), Ip6FreeLinkTxToken(), Ip6SendIcmpError(), and

IP6_LINK_TX_TOKEN::Packet.

Referenced by Ip6CancelFrames(), Ip6CleanService(), Ip6NdFasterTimerTicking(), and Ip6OnArpResolved().

```
EFI_STATUS Ip6InitDADProcess ( IN IP6_INTERFACE *        IpIf,
                               IN IP6_ADDRESS_INFO *     AddressInfo,
                               IN IP6_DAD_CALLBACK       Callback,
                               IN VOID *                 Context
                             )
```

Create a DAD (Duplicate Address Detection) entry and queue it to be performed.

Parameters:

[in]	*IpIf*	Points to the IP6_INTERFACE.
[in]	*AddressInfo*	The address information which needs DAD performed.
[in]	*Callback*	The callback routine that will be called after DAD is performed. This is an optional parameter that may be NULL.
[in]	*Context*	The opaque parameter for a DAD callback routine. This is an optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The DAD entry was created and queued.
EFI_OUT_OF_RESOURCES	Failed to allocate the memory to complete the operation.

References _IP6_DAD_ENTRY::AddressInfo, _IP6_DAD_ENTRY::Callback, _IP6_DAD_ENTRY::Context, _IP6_CONFIG_INSTANCE::DadXmits, _IP6_DAD_ENTRY::Destination, IP6_DAD_ENTRY_SIGNATURE, IP6_INTERFACE_SIGNATURE, IP6_MAX_RTR_SOLICITATION_DELAY, IP6_TIMER_INTERVAL_IN_MS, _IP6_SERVICE::Ip6ConfigInstance, Ip6CreateSNMulticastAddr(), Ip6FindDADEntry(), Ip6JoinGroup(), Ip6OnDADFinished(), _IP6_DAD_ENTRY::Link, _IP6_DAD_ENTRY::MaxTransmit, _IP6_DAD_ENTRY::Receive, _IP6_DAD_ENTRY::RetransTick, _IP6_DAD_ENTRY::Signature, and _IP6_DAD_ENTRY::Transmit.

Referenced by Ip6NdFasterTimerTicking().

```
VOID EFIAPI Ip6NdFasterTimerTicking ( IN EFI_EVENT   Event,
                                      IN VOID *      Context
                                    )
```

The heartbeat timer of ND module in IP6_TIMER_INTERVAL_IN_MS milliseconds. This time routine handles DAD module and neighbor state transition. It is also responsible for sending out router solicitations.

Parameters:

[in]	*Event*	The IP6 service instance's heartbeat timer.
[in]	*Context*	The IP6 service instance.

References _IP6_ADDRESS_INFO::Address, _IP6_DAD_ENTRY::AddressInfo, _IP6_DELAY_JOIN_LIST::AddressInfo, _IP6_DELAY_JOIN_LIST::Context, _IP6_DELAY_JOIN_LIST::DadCallback, _IP6_INTERFACE::DelayJoinList, _IP6_DELAY_JOIN_LIST::DelayTime, _IP6_DAD_ENTRY::Destination, _IP6_INTERFACE::DupAddrDetectList, _IP6_NEIGHBOR_ENTRY::Dynamic, _IP6_DELAY_JOIN_LIST::Interface, _IP6_SERVICE::Interfaces, IP6_GET_TICKS,

IP6_INFINIT_LIFETIME, IP6_MAX_UNICAST_SOLICIT, IP6_RTR_SOLICITATION_INTERVAL, IP6_SERVICE_SIGNATURE, _IP6_SERVICE::Ip6ConfigInstance, Ip6CreateSNMulticastAddr(), Ip6FreeNeighborEntry(), Ip6InitDADProcess(), Ip6OnDADFinished(), Ip6SelectSourceAddress(), Ip6SendNeighborSolicit(), Ip6SendRouterSolicit(), _IP6_DELAY_JOIN_LIST::Link, _IP6_DAD_ENTRY::MaxTransmit, _IP6_NEIGHBOR_ENTRY::Neighbor, _IP6_SERVICE::NeighborTable, _IP6_CONFIG_INSTANCE::Policy, _IP6_DAD_ENTRY::Receive, _IP6_DAD_ENTRY::RetransTick, _IP6_SERVICE::RetransTimer, _IP6_SERVICE::RouterAdvertiseReceived, _IP6_SERVICE::SnpMode, _IP6_SERVICE::SolicitTimer, _IP6_NEIGHBOR_ENTRY::State, _IP6_NEIGHBOR_ENTRY::Ticks, _IP6_SERVICE::Ticks, _IP6_NEIGHBOR_ENTRY::Transmit, and _IP6_DAD_ENTRY::Transmit.

Referenced by Ip6CreateService().

VOID Ip6NdTimerTicking (IN IP6_SERVICE * *IpSb*)

The heartbeat timer of ND module in 1 second. This time routine handles following things: 1) maitain default router list; 2) maintain prefix options; 3) maintain route caches.

Parameters:

`[in]` *IpSb* The IP6 service binding instance.

References IP6_INF_ROUTER_LIFETIME, IP6_INFINIT_LIFETIME, IP6_ROUTE_CACHE_HASH_SIZE, IP6_ROUTE_CACHE_MAX, Ip6DestroyDefaultRouter(), Ip6DestroyPrefixListEntry(), Ip6FreeRouteCacheEntry(), _IP6_DEFAULT_ROUTER::Lifetime, _IP6_PREFIX_LIST_ENTRY::PreferredLifetime, and _IP6_PREFIX_LIST_ENTRY::ValidLifetime.

Referenced by Ip6TimerTicking().

VOID Ip6OnArpResolved (IN VOID * *Context*)

Callback function when address resolution is finished. It will cancel all the queued frames if the address resolution failed, or transmit them if the request succeeded.

Parameters:

`[in]` *Context* The context of the callback, a pointer to IP6_NEIGHBOR_ENTRY.

References IP6_LINK_TX_TOKEN::CallBack, IP6_LINK_TX_TOKEN::Context, IP6_LINK_TX_TOKEN::DstMac, _IP6_NEIGHBOR_ENTRY::Frames, _IP6_NEIGHBOR_ENTRY::Interface, IP6_DELAY_FIRST_PROBE_TIME, IP6_GET_TICKS, IP6_SERVICE_SIGNATURE, Ip6FreeLinkTxToken(), Ip6FreeNeighborEntry(), IP6_LINK_TX_TOKEN::Link, _IP6_NEIGHBOR_ENTRY::LinkAddress, _IP6_SERVICE::Mnp, IP6_LINK_TX_TOKEN::MnpToken, mZeroMacAddress, IP6_LINK_TX_TOKEN::Packet, _IP6_INTERFACE::SentFrames, _IP6_INTERFACE::Service, _IP6_SERVICE::Signature, _IP6_SERVICE::SnpMode, _IP6_NEIGHBOR_ENTRY::State, and _IP6_NEIGHBOR_ENTRY::Ticks.

Referenced by Ip6AddNeighbor(), Ip6Output(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), Ip6ProcessRouterAdvertise(), and Ip6SendNeighborSolicit().

VOID Ip6OnDADFinished (IN BOOLEAN *IsDadPassed*,
IN IP6_INTERFACE * *IpIf*,
IN IP6_DAD_ENTRY * *DadEntry*
)

The function to be called after DAD (Duplicate Address Detection) is performed.

Parameters:

[in]	*IsDadPassed*	If TRUE, the DAD operation succeed. Otherwise, the DAD operation failed.
[in]	*IpIf*	Points to the IP6_INTERFACE.
[in]	*DadEntry*	The DAD entry which already performed DAD.

References _IP6_ADDRESS_INFO::Address, _IP6_CONFIG_INSTANCE::Dhcp6, _IP6_CONFIG_INSTANCE::Dhcp6Event, _IP6_SERVICE::Dhcp6NeedInfoRequest, _IP6_SERVICE::Dhcp6NeedStart, _IP6_SERVICE::FasterTimer, IP6_CONFIG_DHCP6_OPTION_DNS_SERVERS, IP6_CONFIG_DHCP6_OPTION_ORO, IP6_INFINIT_LIFETIME, IP6_LINK_LOCAL_PREFIX_LENGTH, IP6_LINK_LOCAL_SCOPE, Ip6AddAddr(), _IP6_SERVICE::Ip6ConfigInstance, Ip6ConfigOnDhcp6Reply(), Ip6CreatePrefixListEntry(), Ip6LeaveGroup(), Ip6SetToAllNodeMulticast(), _IP6_SERVICE::LinkLocalAddr, _IP6_SERVICE::LinkLocalDadFail, _IP6_SERVICE::LinkLocalOk, _IP6_SERVICE::Mnp, _IP6_ADDRESS_INFO::PreferredLifetime, _IP6_ADDRESS_INFO::PrefixLength, _IP6_SERVICE::Timer, and _IP6_ADDRESS_INFO::ValidLifetime.

Referenced by Ip6ConfigOnPolicyChanged(), Ip6InitDADProcess(), Ip6NdFasterTimerTicking(), Ip6ProcessNeighborAdvertise(), and Ip6ProcessNeighborSolicit().

```
EFI_STATUS Ip6ProcessNeighborAdvertise ( IN IP6_SERVICE *        IpSb,
                                         IN EFI_IP6_HEADER *     Head,
                                         IN NET_BUF *            Packet
                                       )
```

Process the Neighbor Advertisement message.

Parameters:

[in]	*IpSb*	The IP service that received the packet.
[in]	*Head*	The IP head of the message.
[in]	*Packet*	The content of the message with IP head removed.

Return values:

EFI_SUCCESS	The packet processed successfully.
EFI_INVALID_PARAMETER	The packet is invalid.
EFI_ICMP_ERROR	The packet indicates that DAD is failed.
Others	Failed to process the packet.

References _IP6_NEIGHBOR_ENTRY::CallBack, _IP6_ETHE_ADDR_OPTION::EtherAddr, IP6_GET_TICKS, IP6_HOP_LIMIT, IP6_INFINIT_LIFETIME, IP6_IS_ROUTER_FLAG, IP6_ND_LENGTH, IP6_OVERRIDE_FLAG, IP6_SOLICITED_FLAG, Ip6DestroyDefaultRouter(), Ip6FindDADEntry(), Ip6FindDefaultRouter(), Ip6FindNeighborEntry(), Ip6IsNDOptionValid(), Ip6OnDADFinished(), Ip6OptionEtherTarget, _IP6_NEIGHBOR_ENTRY::IsRouter, _IP6_NEIGHBOR_ENTRY::LinkAddress, _IP6_DEFAULT_ROUTER::NeighborCache, _IP6_NEIGHBOR_ENTRY::State, and _IP6_NEIGHBOR_ENTRY::Ticks.

Referenced by Ip6ProcessIcmpInformation().

```
EFI_STATUS Ip6ProcessNeighborSolicit ( IN IP6_SERVICE *        IpSb,
                                       IN EFI_IP6_HEADER *     Head,
```

```
                                    IN NET_BUF *          Packet
                                )
```

Process the Neighbor Solicitation message. The message may be sent for Duplicate Address Detection or Address Resolution.

Parameters:

[in]	*IpSb*	The IP service that received the packet.
[in]	*Head*	The IP head of the message.
[in]	*Packet*	The content of the message with IP head removed.

Return values:

EFI_SUCCESS	The packet processed successfully.
EFI_INVALID_PARAMETER	The packet is invalid.
EFI_ICMP_ERROR	The packet indicates that DAD is failed.
Others	Failed to process the packet.

References _IP6_NEIGHBOR_ENTRY::CallBack, _IP6_ETHE_ADDR_OPTION::EtherAddr, IP6_HOP_LIMIT, IP6_INFINIT_LIFETIME, IP6_LINK_LOCAL_SCOPE, IP6_ND_LENGTH, Ip6CreateNeighborEntry(), Ip6FindDADEntry(), Ip6FindNeighborEntry(), Ip6IsNDOptionValid(), Ip6IsOneOfSetAddress(), Ip6IsSNMulticastAddr(), Ip6OnArpResolved(), Ip6OnDADFinished(), Ip6OptionEtherSource, Ip6SendNeighborAdvertise(), Ip6SetToAllNodeMulticast(), _IP6_NEIGHBOR_ENTRY::LinkAddress, _IP6_DAD_ENTRY::Receive, _IP6_NEIGHBOR_ENTRY::State, and _IP6_NEIGHBOR_ENTRY::Ticks.

Referenced by Ip6ProcessIcmpInformation().

```
EFI_STATUS Ip6ProcessRedirect ( IN IP6_SERVICE *        IpSb,
                                IN EFI_IP6_HEADER *     Head,
                                IN NET_BUF *            Packet
                              )
```

Process the ICMPv6 redirect message. Find the instance, then update its route cache.

Parameters:

[in]	*IpSb*	The IP6 service binding instance that received the packet.
[in]	*Head*	The IP head of the received ICMPv6 packet.
[in]	*Packet*	The content of the ICMPv6 redirect packet with the IP head removed.

Return values:

EFI_INVALID_PARAMETER	The parameter is invalid.
EFI_OUT_OF_RESOURCES	Insuffcient resources to complete the operation.
EFI_SUCCESS	Successfully updated the route caches.

References _IP6_NEIGHBOR_ENTRY::CallBack, _IP6_ETHE_ADDR_OPTION::EtherAddr, IP6_ROUTE_ENTRY::Flag, IP6_DIRECT_ROUTE, IP6_HOP_LIMIT, IP6_INFINIT_LIFETIME, IP6_REDITECT_LENGTH, IP6_ROUTE_CACHE_HASH, Ip6CreateNeighborEntry(), Ip6CreateRouteCacheEntry(), Ip6CreateRouteEntry(), Ip6FindNeighborEntry(), Ip6FindRouteCache(), Ip6FindRouteEntry(), Ip6IsNDOptionValid(), Ip6IsOneOfSetAddress(), Ip6OnArpResolved(), Ip6OptionEtherTarget, _IP6_NEIGHBOR_ENTRY::IsRouter, _IP6_ETHE_ADDR_OPTION::Length, IP6_ROUTE_CACHE_ENTRY::Link, _IP6_NEIGHBOR_ENTRY::LinkAddress, IP6_ROUTE_CACHE_ENTRY::NextHop, _IP6_NEIGHBOR_ENTRY::State, IP6_ROUTE_CACHE_ENTRY::Tag, and _IP6_NEIGHBOR_ENTRY::Ticks.

Referenced by Ip6ProcessIcmpInformation().

```
EFI_STATUS Ip6ProcessRouterAdvertise ( IN IP6_SERVICE *        IpSb,
                                       IN EFI_IP6_HEADER *     Head,
                                       IN NET_BUF *            Packet
                                     )
```

Process the Router Advertisement message according to RFC4861.

Parameters:

[in]	*IpSb*	The IP service that received the packet.
[in]	*Head*	The IP head of the message.
[in]	*Packet*	The content of the message with the IP head removed.

Return values:

EFI_SUCCESS	The packet processed successfully.
EFI_INVALID_PARAMETER	The packet is invalid.
EFI_OUT_OF_RESOURCES	Insufficient resources to complete the operation.
Others	Failed to process the packet.

References _IP6_NEIGHBOR_ENTRY::CallBack, IP6_AUTO_CONFIG_FLAG, IP6_HOP_LIMIT, IP6_INFINIT_LIFETIME, IP6_LINK_LOCAL_PREFIX_LENGTH, IP6_M_ADDR_CONFIG_FLAG, IP6_O_CONFIG_FLAG, IP6_ON_LINK_FLAG, IP6_RA_LENGTH, Ip6ConfigStartStatefulAutoConfig(), Ip6CreateDefaultRouter(), Ip6CreateNeighborEntry(), Ip6CreatePrefixListEntry(), Ip6DestroyDefaultRouter(), Ip6DestroyPrefixListEntry(), Ip6FindDADEntry(), Ip6FindDefaultRouter(), Ip6FindNeighborEntry(), Ip6FindPrefixListEntry(), Ip6IsNDOptionValid(), Ip6IsOneOfSetAddress(), Ip6OnArpResolved(), Ip6OptionEtherSource, Ip6OptionMtu, Ip6OptionPrefixInfo, Ip6SetAddress(), Ip6UpdateReachableTime(), _IP6_NEIGHBOR_ENTRY::IsRouter, _IP6_DEFAULT_ROUTER::Lifetime, _IP6_NEIGHBOR_ENTRY::LinkAddress, _IP6_DEFAULT_ROUTER::NeighborCache, _IP6_PREFIX_LIST_ENTRY::PreferredLifetime, _IP6_NEIGHBOR_ENTRY::State, _IP6_NEIGHBOR_ENTRY::Ticks, and _IP6_PREFIX_LIST_ENTRY::ValidLifetime.

Referenced by Ip6ProcessIcmpInformation().

```
EFI_STATUS Ip6SendNeighborAdvertise ( IN IP6_SERVICE *          IpSb,
                                      IN EFI_IPv6_ADDRESS *     SourceAddress,
                                      IN EFI_IPv6_ADDRESS *     DestinationAddress,
                                      IN EFI_IPv6_ADDRESS *     TargetIp6Address,
                                      IN EFI_MAC_ADDRESS *      TargetLinkAddress,
                                      IN BOOLEAN                IsRouter,
                                      IN BOOLEAN                Override,
                                      IN BOOLEAN                Solicited
                                    )
```

Generate a Neighbor Advertisement message and send it out to Destination Address.

Parameters:

[in]	*IpSb*	The IP service to send the packet.
[in]	*SourceAddress*	The source address of the message.
[in]	*DestinationAddress*	The destination address of the message.
[in]	*TargetIp6Address*	

		The target address field in the Neighbor Solicitation message that prompted this advertisement.
[in]	*TargetLinkAddress*	The MAC address for the target, i.e. the sender of the advertisement.
[in]	*IsRouter*	If TRUE, indicates the sender is a router.
[in]	*Override*	If TRUE, indicates the advertisement should override an existing cache entry and update the MAC address.
[in]	*Solicited*	If TRUE, indicates the advertisement was sent in response to a Neighbor Solicitation from the Destination address.

Return values:

EFI_OUT_OF_RESOURCES	Insufficient resources to complete the operation.
EFI_SUCCESS	The Neighbor Advertise message was successfully sent.

References _IP6_ETHE_ADDR_OPTION::EtherAddr, IP6_HOP_LIMIT, IP6_IS_ROUTER_FLAG, IP6_OVERRIDE_FLAG, IP6_SERVICE_SIGNATURE, IP6_SOLICITED_FLAG, Ip6OptionEtherTarget, Ip6Output(), Ip6SysPacketSent(), _IP6_ETHE_ADDR_OPTION::Length, and _IP6_ETHE_ADDR_OPTION::Type.

Referenced by Ip6ProcessNeighborSolicit().

```
EFI_STATUS Ip6SendNeighborSolicit ( IN IP6_SERVICE *          IpSb,
                                    IN EFI_IPv6_ADDRESS *     SourceAddress,
                                    IN EFI_IPv6_ADDRESS *     DestinationAddress,
                                    IN EFI_IPv6_ADDRESS *     TargetIp6Address,
                                    IN EFI_MAC_ADDRESS *      SourceLinkAddress
                                  )
```

Generate the Neighbor Solicitation message and send it to the Destination Address.

Parameters:

[in]	*IpSb*	The IP service to send the packet
[in]	*SourceAddress*	The source address of the message.
[in]	*DestinationAddress*	The destination address of the message.
[in]	*TargetIp6Address*	The IP address of the target of the solicitation. It must not be a multicast address.
[in]	*SourceLinkAddress*	The MAC address for the sender. If not NULL, a source link-layer address option will be appended to the message.

Return values:

EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_OUT_OF_RESOURCES	Insufficient resources to complete the operation.
EFI_SUCCESS	The Neighbor Advertise message was successfully sent.

References _IP6_ETHE_ADDR_OPTION::EtherAddr, IP6_HOP_LIMIT, IP6_SERVICE_SIGNATURE, Ip6CreateNeighborEntry(), Ip6FindNeighborEntry(), Ip6IsSNMulticastAddr(), Ip6OnArpResolved(), Ip6OptionEtherSource, Ip6Output(), Ip6SysPacketSent(), _IP6_ETHE_ADDR_OPTION::Length, and _IP6_ETHE_ADDR_OPTION::Type.

Referenced by Ip6NdFasterTimerTicking(), and Ip6Output().

```
EFI_STATUS Ip6SendRouterSolicit ( IN IP6_SERVICE *          IpSb,
```

```
    IN IP6_INTERFACE *          Interface,
    IN EFI_IPv6_ADDRESS *       SourceAddress,
    IN EFI_IPv6_ADDRESS *       DestinationAddress,
    IN EFI_MAC_ADDRESS *        SourceLinkAddress
  )
```

Generate router solicit message and send it out to Destination Address or All Router Link Local scope multicast address.

Parameters:

[in]	*IpSb*	The IP service to send the packet.
[in]	*Interface*	If not NULL, points to the IP6 interface to send the packet.
[in]	*SourceAddress*	If not NULL, the source address of the message.
[in]	*DestinationAddress*	If not NULL, the destination address of the message.
[in]	*SourceLinkAddress*	If not NULL, the MAC address of the source. A source link-layer address option will be appended to the message.

Return values:

EFI_OUT_OF_RESOURCES	Insufficient resources to complete the operation.
EFI_SUCCESS	The router solicit message was successfully sent.

References _IP6_ETHE_ADDR_OPTION::EtherAddr, IP6_HOP_LIMIT, IP6_LINK_LOCAL_SCOPE, IP6_SERVICE_SIGNATURE, Ip6OptionEtherSource, Ip6Output(), Ip6SetToAllNodeMulticast(), Ip6SysPacketSent(), _IP6_ETHE_ADDR_OPTION::Length, and _IP6_ETHE_ADDR_OPTION::Type.

Referenced by Ip6NdFasterTimerTicking().

VOID Ip6UpdateReachableTime (IN OUT IP6_SERVICE * *IpSb*)

Update the ReachableTime in IP6 service binding instance data, in milliseconds.

Parameters:

[in,out]	*IpSb*	Points to the IP6_SERVICE.

References IP6_MIN_RANDOM_FACTOR_SCALED, and IP6_RANDOM_FACTOR_SCALE.

Referenced by Ip6CreateService(), and Ip6ProcessRouterAdvertise().

Variable Documentation

EFI_MAC_ADDRESS mZeroMacAddress

Referenced by Ip6CreateNeighborEntry(), and Ip6OnArpResolved().

NetworkPkg/Ip6Dxe/Ip6Nd.h File Reference

Data Structures

Defines

"memItemRight" valign="bottom">IP6_GET_TICKS(Ms) (((Ms) + IP6_TIMER_INTERVAL_IN_MS - 1) / IP6_T

Typedefs

typede

typed
typedef struc
typedef struc
typedef struct

Enumerations

Functions

IP6_DE

IP6_DE

IF

IF

Detailed Description

Definition of Neighbor Discovery support routines.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"paramtype">Ms) (((Ms) + IP6_TIMER_INTERVAL_IN_MS - 1) / IP6_TIMER_INTERVAL_IN_MS)

Referenced by Ip6AddNeighbor(), Ip6ConfigOnPolicyChanged(), Ip6NdFasterTimerTicking(), Ip6OnArpResolved(), Ip6Output(), Ip6ProcessNeighborAdvertise(), and Ip6SendFrame().

Typedef Documentation

typedef VOID(* IP6_ARP_CALLBACK)(VOID *Context)

typedef VOID(* IP6_DAD_CALLBACK)(IN BOOLEAN IsDadPassed, IN EFI_IPv6_ADDRESS *TargetAddress, IN VOID *Context)

typedef struct _IP6_DAD_ENTRY IP6_DAD_ENTRY

typedef struct _IP6_DEFAULT_ROUTER IP6_DEFAULT_ROUTER

typedef struct _IP6_DELAY_JOIN_LIST IP6_DELAY_JOIN_LIST

typedef struct _IP6_ETHE_ADDR_OPTION IP6_ETHER_ADDR_OPTION

typedef struct _IP6_MTU_OPTION IP6_MTU_OPTION

typedef struct _IP6_NEIGHBOR_ENTRY IP6_NEIGHBOR_ENTRY

typedef struct _IP6_PREFIX_INFO_OPTION IP6_PREFIX_INFO_OPTION

typedef struct _IP6_PREFIX_LIST_ENTRY IP6_PREFIX_LIST_ENTRY

Enumeration Type Documentation

anonymous enum

Enumerator:

IP6_INF_ROUTER_LIFETIME
IP6_MAX_RTR_SOLICITATION_DELAY 1000 milliseconds
IP6_MAX_RTR_SOLICITATIONS
IP6_RTR_SOLICITATION_INTERVAL
IP6_MIN_RANDOM_FACTOR_SCALED
IP6_MAX_RANDOM_FACTOR_SCALED
IP6_RANDOM_FACTOR_SCALE
IP6_MAX_MULTICAST_SOLICIT

IP6_MAX_UNICAST_SOLICIT
IP6_MAX_ANYCAST_DELAY_TIME
IP6_MAX_NEIGHBOR_ADV
IP6_REACHABLE_TIME
IP6_RETRANS_TIMER
IP6_DELAY_FIRST_PROBE_TIME
IP6_MIN_LINK_MTU
IP6_MAX_LINK_MTU
IP6_IS_ROUTER_FLAG
IP6_SOLICITED_FLAG
IP6_OVERRIDE_FLAG
IP6_M_ADDR_CONFIG_FLAG
IP6_O_CONFIG_FLAG
IP6_ON_LINK_FLAG
IP6_AUTO_CONFIG_FLAG
IP6_ND_LENGTH
IP6_RA_LENGTH
IP6_REDITECT_LENGTH
IP6_DAD_ENTRY_SIGNATURE

Function Documentation

```
EFI_STATUS Ip6AddNeighbor ( IN IP6_SERVICE *          IpSb,
                            IN EFI_IPv6_ADDRESS *     TargetIp6Address,
                            IN EFI_MAC_ADDRESS *      TargetLinkAddress,
                            IN UINT32                 Timeout,
                            IN BOOLEAN                Override
                          )
```

Add Neighbor cache entries. It is a work function for EfiIp6Neighbors().

Parameters:

[in]	*IpSb*	The IP6 service binding instance.
[in]	*TargetIp6Address*	Pointer to Target IPv6 address.
[in]	*TargetLinkAddress*	Pointer to link-layer address of the target. Ignored if NULL.
[in]	*Timeout*	Time in 100-ns units that this entry will remain in the neighbor cache. It will be deleted after Timeout. A value of zero means that the entry is permanent. A non-zero value means that the entry is dynamic.
[in]	*Override*	If TRUE, the cached link-layer address of the matching entry will be overridden and updated; if FALSE, and if a corresponding cache entry already existed, EFI_ACCESS_DENIED will be returned.

Return values:

EFI_SUCCESS	The neighbor cache entry has been added.
EFI_OUT_OF_RESOURCES	Could not add the entry to the neighbor cache due to insufficient resources.
EFI_NOT_FOUND	TargetLinkAddress is NULL.
EFI_ACCESS_DENIED	The to-be-added entry is already defined in the neighbor cache, and that entry is tagged as un-overridden (when DeleteFlag is FALSE).

References _IP6_NEIGHBOR_ENTRY::Dynamic, IP6_GET_TICKS, IP6_INFINIT_LIFETIME, Ip6CreateNeighborEntry(), Ip6FindNeighborEntry(), Ip6OnArpResolved(), _IP6_NEIGHBOR_ENTRY::LinkAddress, _IP6_NEIGHBOR_ENTRY::State, and _IP6_NEIGHBOR_ENTRY::Ticks.

Referenced by EfiIp6Neighbors().

```
EFI_STATUS Ip6BuildEfiNeighborCache ( IN IP6_PROTOCOL *                  IpInstance,
                                      OUT UINT32 *                       NeighborCount,
                                      OUT EFI_IP6_NEIGHBOR_CACHE **      NeighborCache
                                    )
```

Build a array of EFI_IP6_NEIGHBOR_CACHE to be returned to the caller. The number of EFI_IP6_NEIGHBOR_CACHE is also returned.

Parameters:

[in]	*IpInstance*	The pointer to IP6_PROTOCOL instance.
[out]	*NeighborCount*	The number of returned neighbor cache entries.
[out]	*NeighborCache*	The pointer to the array of EFI_IP6_NEIGHBOR_CACHE.

Return values:

EFI_SUCCESS	The EFI_IP6_NEIGHBOR_CACHE successfully built.
EFI_OUT_OF_RESOURCES	Failed to allocate the memory for the route table.

References IP6_PROTOCOL_SIGNATURE, _IP6_NEIGHBOR_ENTRY::LinkAddress, _IP6_NEIGHBOR_ENTRY::Neighbor, _IP6_SERVICE::NeighborTable, and _IP6_NEIGHBOR_ENTRY::State.

Referenced by EfiIp6GetModeData().

```
EFI_STATUS Ip6BuildPrefixTable ( IN IP6_PROTOCOL *                 IpInstance,
                                 OUT UINT32 *                      PrefixCount,
                                 OUT EFI_IP6_ADDRESS_INFO **       PrefixTable
                               )
```

Build a array of EFI_IP6_ADDRESS_INFO to be returned to the caller. The number of prefix entries is also returned.

Parameters:

[in]	*IpInstance*	The pointer to IP6_PROTOCOL instance.
[out]	*PrefixCount*	The number of returned prefix entries.
[out]	*PrefixTable*	The pointer to the array of PrefixTable.

Return values:

EFI_SUCCESS	The prefix table successfully built.
EFI_OUT_OF_RESOURCES	Failed to allocate the memory for the prefix table.

References IP6_PROTOCOL_SIGNATURE, _IP6_SERVICE::OnlinkPrefix, _IP6_PREFIX_LIST_ENTRY::Prefix, and _IP6_PREFIX_LIST_ENTRY::PrefixLength.

Referenced by EfiIp6GetModeData().

VOID Ip6CleanDefaultRouterList (IN IP6_SERVICE * *IpSb*)

Clean an IP6 default router list.

Parameters:

[in] *IpSb* The pointer to the IP6_SERVICE instance.

References Ip6DestroyDefaultRouter().

Referenced by Ip6CleanService(), and Ip6ConfigOnPolicyChanged().

VOID Ip6CleanPrefixListTable (IN IP6_SERVICE * *IpSb*,
IN LIST_ENTRY * *ListHead*
)

Release the resource in prefix list table, and destroy the list entry and corresponding addresses or route entries.

Parameters:

[in] *IpSb* The pointer to the IP6_SERVICE instance.
[in] *ListHead* The list entry head of the prefix list table.

Release the resource in the prefix list table, and destroy the list entry and corresponding addresses or route entries.

Parameters:

[in] *IpSb* The pointer to the IP6_SERVICE instance.
[in] *ListHead* The list entry head of the prefix list table.

References Ip6DestroyPrefixListEntry().

Referenced by Ip6CleanService(), and Ip6ConfigOnPolicyChanged().

IP6_DEFAULT_ROUTER* Ip6CreateDefaultRouter (IN IP6_SERVICE * *IpSb*,
IN EFI_IPv6_ADDRESS * *Ip6Address*,
IN UINT16 *RouterLifetime*
)

Allocate and initialize an IP6 default router entry.

Parameters:

[in] *IpSb* The pointer to the IP6_SERVICE instance.
[in] *Ip6Address* The IPv6 address of the default router.
[in] *RouterLifetime* The lifetime associated with the default router, in units of seconds.

Returns:

NULL if it failed to allocate memory for the default router node. Otherwise, point to the created default router node.

References IP6_SERVICE_SIGNATURE, Ip6CreateRouteEntry(), Ip6FindNeighborEntry(), _IP6_DEFAULT_ROUTER::Lifetime, _IP6_DEFAULT_ROUTER::Link, IP6_ROUTE_ENTRY::Link, _IP6_DEFAULT_ROUTER::NeighborCache, _IP6_DEFAULT_ROUTER::RefCnt, and _IP6_DEFAULT_ROUTER::Router.

Referenced by Ip6ConfigSetGateway(), and Ip6ProcessRouterAdvertise().

```
IP6_NEIGHBOR_ENTRY* Ip6CreateNeighborEntry ( IN IP6_SERVICE *          IpSb,
                                             IN IP6_ARP_CALLBACK       CallBack,
                                             IN EFI_IPv6_ADDRESS *     Ip6Address,
                                             IN EFI_MAC_ADDRESS *      LinkAddress
                                           )
```

Allocate and initialize an IP6 neighbor cache entry.

Parameters:

[in]	*IpSb*	The pointer to the IP6_SERVICE instance.
[in]	*CallBack*	The callback function to be called when address resolution is finished.
[in]	*Ip6Address*	Points to the IPv6 address of the neighbor.
[in]	*LinkAddress*	Points to the MAC address of the neighbor. Ignored if NULL.

Returns:

NULL if failed to allocate memory for the neighbor cache entry. Otherwise, point to the created neighbor cache entry.

References _IP6_NEIGHBOR_ENTRY::ArpFree, _IP6_NEIGHBOR_ENTRY::CallBack, _IP6_NEIGHBOR_ENTRY::Dynamic, _IP6_NEIGHBOR_ENTRY::Frames, _IP6_NEIGHBOR_ENTRY::Interface, IP6_MAX_MULTICAST_SOLICIT, IP6_SERVICE_SIGNATURE, Ip6FindDefaultRouter(), _IP6_NEIGHBOR_ENTRY::IsRouter, _IP6_NEIGHBOR_ENTRY::Link, _IP6_NEIGHBOR_ENTRY::LinkAddress, mZeroMacAddress, _IP6_NEIGHBOR_ENTRY::Neighbor, _IP6_DEFAULT_ROUTER::NeighborCache, _IP6_NEIGHBOR_ENTRY::RefCnt, _IP6_NEIGHBOR_ENTRY::State, and _IP6_NEIGHBOR_ENTRY::Transmit.

Referenced by Ip6AddNeighbor(), Ip6Output(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), Ip6ProcessRouterAdvertise(), and Ip6SendNeighborSolicit().

```
IP6_PREFIX_LIST_ENTRY* Ip6CreatePrefixListEntry ( IN IP6_SERVICE *          IpSb,
                                                  IN BOOLEAN                OnLinkOrAuto,
                                                  IN UINT32                 ValidLifetime,
                                                  IN UINT32                 PreferredLifetime,
                                                  IN UINT8                  PrefixLength,
                                                  IN EFI_IPv6_ADDRESS *     Prefix
                                                )
```

Allocate and initialize a IP6 prefix list entry.

Parameters:

[in]	*IpSb*	The pointer to IP6_SERVICE instance.
[in]	*OnLinkOrAuto*	If TRUE, the entry is created for the on link prefix list. Otherwise, it is created for the autoconfiguration prefix list.
[in]	*ValidLifetime*	The length of time in seconds that the prefix is valid for the purpose of on-link determination.
[in]	*PreferredLifetime*	The length of time in seconds that addresses generated from the prefix via stateless address autoconfiguration remain preferred.
[in]	*PrefixLength*	The prefix length of the Prefix.
[in]	*Prefix*	The prefix address.

Returns:

NULL if it failed to allocate memory for the prefix node. Otherwise, point to the created or existing prefix list entry.

References IP6_ROUTE_ENTRY::Flag, IP6_DIRECT_ROUTE, IP6_SERVICE_SIGNATURE, Ip6CreateRouteEntry(), Ip6FindPrefixListEntry(), _IP6_PREFIX_LIST_ENTRY::Link, IP6_ROUTE_ENTRY::Link, _IP6_PREFIX_LIST_ENTRY::PreferredLifetime, _IP6_PREFIX_LIST_ENTRY::Prefix, _IP6_PREFIX_LIST_ENTRY::PrefixLength, _IP6_PREFIX_LIST_ENTRY::RefCnt, and _IP6_PREFIX_LIST_ENTRY::ValidLifetime.

Referenced by Ip6ConfigOnPolicyChanged(), Ip6ConfigSetMaunualAddress(), Ip6OnDADFinished(), and Ip6ProcessRouterAdvertise().

```
EFI_STATUS Ip6DelNeighbor ( IN IP6_SERVICE *          IpSb,
                            IN EFI_IPv6_ADDRESS *     TargetIp6Address,
                            IN EFI_MAC_ADDRESS *      TargetLinkAddress,
                            IN UINT32                 Timeout,
                            IN BOOLEAN                Override
                          )
```

Delete or update Neighbor cache entries. It is a work function for EfiIp6Neighbors().

Parameters:

[in]	*IpSb*	The IP6 service binding instance.
[in]	*TargetIp6Address*	Pointer to Target IPv6 address.
[in]	*TargetLinkAddress*	Pointer to link-layer address of the target. Ignored if NULL.
[in]	*Timeout*	Time in 100-ns units that this entry will remain in the neighbor cache. It will be deleted after Timeout. A value of zero means that the entry is permanent. A non-zero value means that the entry is dynamic.
[in]	*Override*	If TRUE, the cached link-layer address of the matching entry will be overridden and updated; if FALSE, and if a corresponding cache entry already existed, EFI_ACCESS_DENIED will be returned.

Return values:

EFI_SUCCESS	The neighbor cache entry has been updated or deleted.
EFI_NOT_FOUND	This entry is not in the neighbor cache.

References Ip6FindNeighborEntry(), and _IP6_NEIGHBOR_ENTRY::Link.

Referenced by EfiIp6Neighbors().

```
VOID Ip6DestroyDefaultRouter ( IN IP6_SERVICE *          IpSb,
                               IN IP6_DEFAULT_ROUTER *   DefaultRouter
                             )
```

Destroy an IP6 default router entry.

Parameters:

[in]	*IpSb*	The pointer to the IP6_SERVICE instance.
[in]	*DefaultRouter*	The to be destroyed IP6_DEFAULT_ROUTER.

References Ip6DelRoute().

Referenced by Ip6CleanDefaultRouterList(), Ip6ConfigSetGateway(), Ip6FreeNeighborEntry(), Ip6NdTimerTicking(), Ip6ProcessNeighborAdvertise(), and Ip6ProcessRouterAdvertise().

```
VOID Ip6DestroyPrefixListEntry ( IN IP6_SERVICE *                IpSb,
                                 IN IP6_PREFIX_LIST_ENTRY *      PrefixEntry,
                                 IN BOOLEAN                      OnLinkOrAuto,
                                 IN BOOLEAN                      ImmediateDelete
                               )
```

Destroy a IP6 prefix list entry.

Parameters:

[in]	*IpSb*	The pointer to IP6_SERVICE instance.
[in]	*PrefixEntry*	The to be destroyed prefix list entry.
[in]	*OnLinkOrAuto*	If TRUE, the entry is removed from on link prefix list. Otherwise remove from autoconfiguration prefix list.
[in]	*ImmediateDelete*	If TRUE, remove the entry directly. Otherwise, check the reference count to see whether it should be removed.

References _IP6_INTERFACE::AddressCount, _IP6_INTERFACE::AddressList, IP6_INTERFACE_SIGNATURE, Ip6DelRoute(), and Ip6RemoveAddr().

Referenced by Ip6CleanPrefixListTable(), Ip6ConfigSetMaunualAddress(), Ip6NdTimerTicking(), and Ip6ProcessRouterAdvertise().

```
IP6_DAD_ENTRY* Ip6FindDADEntry ( IN IP6_SERVICE *          IpSb,
                                 IN EFI_IPv6_ADDRESS *     Target,
                                 OUT IP6_INTERFACE **      Interface
                               )
```

Search IP6_DAD_ENTRY from the Duplicate Address Detection List.

Parameters:

[in]	*IpSb*	The pointer to the IP6_SERVICE instance.
[in]	*Target*	The address information which needs DAD performed .
[out]	*Interface*	If not NULL, output the IP6 interface that configures the tentative address.

Returns:

NULL if failed to find the matching DAD entry. Otherwise, point to the found DAD entry.

References _IP6_ADDRESS_INFO::Address, _IP6_DAD_ENTRY::AddressInfo, _IP6_INTERFACE::DupAddrDetectList, and IP6_DAD_ENTRY_SIGNATURE.

Referenced by Ip6InitDADProcess(), Ip6ProcessNeighborAdvertise(), Ip6ProcessNeighborSolicit(), and Ip6ProcessRouterAdvertise().

```
IP6_DEFAULT_ROUTER* Ip6FindDefaultRouter ( IN IP6_SERVICE *          IpSb,
                                           IN EFI_IPv6_ADDRESS *     Ip6Address
                                         )
```

Search a default router node from an IP6 default router list.

Parameters:

[in]	*IpSb*	The pointer to the IP6_SERVICE instance.
[in]	*Ip6Address*	The IPv6 address of the to be searched default router node.

Returns:

NULL if it failed to find the matching default router node. Otherwise, point to the found default router node.

References IP6_SERVICE_SIGNATURE, and _IP6_DEFAULT_ROUTER::Router.

Referenced by Ip6ConfigSetGateway(), Ip6CreateNeighborEntry(), Ip6FreeNeighborEntry(), Ip6ProcessNeighborAdvertise(), and Ip6ProcessRouterAdvertise().

```
IP6_NEIGHBOR_ENTRY* Ip6FindNeighborEntry ( IN IP6_SERVICE *        IpSb,
                                           IN
                                           EFI_IPv6_ADDRESS        Ip6Address
                                           *
                                         )
```

Search a IP6 neighbor cache entry.

Parameters:

[in]	*IpSb*	The pointer to the IP6_SERVICE instance.
[in]	*Ip6Address*	Points to the IPv6 address of the neighbor.

Returns:

NULL if it failed to find the matching neighbor cache entry. Otherwise, point to the found neighbor cache entry.

References IP6_SERVICE_SIGNATURE, and _IP6_NEIGHBOR_ENTRY::Neighbor.

Referenced by Ip6AddNeighbor(), Ip6CreateDefaultRouter(), Ip6DelNeighbor(), Ip6Output(), Ip6ProcessNeighborAdvertise(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), Ip6ProcessRouterAdvertise(), Ip6SendFrame(), and Ip6SendNeighborSolicit().

```
IP6_PREFIX_LIST_ENTRY* Ip6FindPrefixListEntry ( IN IP6_SERVICE *          IpSb,
                                                IN BOOLEAN                OnLinkOrAuto,
                                                IN UINT8                  PrefixLength,
                                                IN EFI_IPv6_ADDRESS *     Prefix
                                              )
```

Search the list array to find an IP6 prefix list entry.

Parameters:

[in]	*IpSb*	The pointer to IP6_SERVICE instance.
[in]	*OnLinkOrAuto*	If TRUE, the search the link prefix list, Otherwise search the autoconfiguration prefix list.
[in]	*PrefixLength*	The prefix length of the Prefix
[in]	*Prefix*	The prefix address.

Returns:

NULL if cannot find the IP6 prefix list entry. Otherwise, return the pointer to the IP6 prefix list entry.

References IP6_SERVICE_SIGNATURE, _IP6_PREFIX_LIST_ENTRY::Prefix, and _IP6_PREFIX_LIST_ENTRY::PrefixLength.

Referenced by Ip6ConfigSetMaunualAddress(), Ip6CreatePrefixListEntry(), Ip6IsAnycast(), and Ip6ProcessRouterAdvertise().

```
EFI_STATUS Ip6FreeNeighborEntry ( IN IP6_SERVICE *            IpSb,
                                  IN IP6_NEIGHBOR_ENTRY *     NeighborCache,
                                  IN BOOLEAN                  SendIcmpError,
                                  IN BOOLEAN                  FullFree,
                                  IN EFI_STATUS               IoStatus,
                                  IN IP6_FRAME_TO_CANCEL      FrameToCancel,
                                  IN VOID *                   Context
                                )
```

Free a IP6 neighbor cache entry and remove all the frames on the address resolution queue that pass the FrameToCancel. That is, either FrameToCancel is NULL, or it returns true for the frame.

Parameters:

[in]	*IpSb*	The pointer to the IP6_SERVICE instance.
[in]	*NeighborCache*	The to be free neighbor cache entry.
[in]	*SendIcmpError*	If TRUE, send out ICMP error.
[in]	*FullFree*	If TRUE, remove the neighbor cache entry. Otherwise remove the pending frames.
[in]	*IoStatus*	The status returned to the cancelled frames' callback function.
[in]	*FrameToCancel*	Function to select which frame to cancel. This is an optional parameter that may be NULL.
[in]	*Context*	Opaque parameter to the FrameToCancel. Ignored if FrameToCancel is NULL.

Return values:

EFI_INVALID_PARAMETER	The input parameter is invalid.
EFI_SUCCESS	The operation finished successfully.

References IP6_LINK_TX_TOKEN::CallBack, IP6_LINK_TX_TOKEN::Context, Ip6DestroyDefaultRouter(), Ip6FindDefaultRouter(), Ip6FreeLinkTxToken(), Ip6SendIcmpError(), and IP6_LINK_TX_TOKEN::Packet.

Referenced by Ip6CancelFrames(), Ip6CleanService(), Ip6NdFasterTimerTicking(), and Ip6OnArpResolved().

```
EFI_STATUS Ip6InitDADProcess ( IN IP6_INTERFACE *        IpIf,
                               IN IP6_ADDRESS_INFO *     AddressInfo,
                               IN IP6_DAD_CALLBACK       Callback,
                               IN VOID *                 Context
                             )
```

Create a DAD (Duplicate Address Detection) entry and queue it to be performed.

Parameters:

[in]	*IpIf*	Points to the IP6_INTERFACE.
[in]	*AddressInfo*	The address information which needs DAD performed.

[in]	*Callback*	The callback routine that will be called after DAD is performed. This is an optional parameter that may be NULL.
[in]	*Context*	The opaque parameter for a DAD callback routine. This is an optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The DAD entry was created and queued.
EFI_OUT_OF_RESOURCES	Failed to allocate the memory to complete the operation.

References _IP6_DAD_ENTRY::AddressInfo, _IP6_DAD_ENTRY::Callback, _IP6_DAD_ENTRY::Context, _IP6_CONFIG_INSTANCE::DadXmits, _IP6_DAD_ENTRY::Destination, IP6_DAD_ENTRY_SIGNATURE, IP6_INTERFACE_SIGNATURE, IP6_MAX_RTR_SOLICITATION_DELAY, IP6_TIMER_INTERVAL_IN_MS, _IP6_SERVICE::Ip6ConfigInstance, Ip6CreateSNMulticastAddr(), Ip6FindDADEntry(), Ip6JoinGroup(), Ip6OnDADFinished(), _IP6_DAD_ENTRY::Link, _IP6_DAD_ENTRY::MaxTransmit, _IP6_DAD_ENTRY::Receive, _IP6_DAD_ENTRY::RetransTick, _IP6_DAD_ENTRY::Signature, and _IP6_DAD_ENTRY::Transmit.

Referenced by Ip6NdFasterTimerTicking().

```
VOID EFIAPI Ip6NdFasterTimerTicking ( IN EFI_EVENT  Event,
                                      IN VOID *     Context
                                    )
```

The heartbeat timer of ND module in IP6_TIMER_INTERVAL_IN_MS milliseconds. This time routine handles DAD module and neighbor state transition. It is also responsible for sending out router solicitations.

Parameters:

[in]	*Event*	The IP6 service instance's heartbeat timer.
[in]	*Context*	The IP6 service instance.

References _IP6_ADDRESS_INFO::Address, _IP6_DAD_ENTRY::AddressInfo, _IP6_DELAY_JOIN_LIST::AddressInfo, _IP6_DELAY_JOIN_LIST::Context, _IP6_DELAY_JOIN_LIST::DadCallback, _IP6_INTERFACE::DelayJoinList, _IP6_DELAY_JOIN_LIST::DelayTime, _IP6_DAD_ENTRY::Destination, _IP6_INTERFACE::DupAddrDetectList, _IP6_NEIGHBOR_ENTRY::Dynamic, _IP6_DELAY_JOIN_LIST::Interface, _IP6_SERVICE::Interfaces, IP6_GET_TICKS, IP6_INFINIT_LIFETIME, IP6_MAX_UNICAST_SOLICIT, IP6_RTR_SOLICITATION_INTERVAL, IP6_SERVICE_SIGNATURE, _IP6_SERVICE::Ip6ConfigInstance, Ip6CreateSNMulticastAddr(), Ip6FreeNeighborEntry(), Ip6InitDADProcess(), Ip6OnDADFinished(), Ip6SelectSourceAddress(), Ip6SendNeighborSolicit(), Ip6SendRouterSolicit(), _IP6_DELAY_JOIN_LIST::Link, _IP6_DAD_ENTRY::MaxTransmit, _IP6_NEIGHBOR_ENTRY::Neighbor, _IP6_SERVICE::NeighborTable, _IP6_CONFIG_INSTANCE::Policy, _IP6_DAD_ENTRY::Receive, _IP6_DAD_ENTRY::RetransTick, _IP6_SERVICE::RetransTimer, _IP6_SERVICE::RouterAdvertiseReceived, _IP6_SERVICE::SnpMode, _IP6_SERVICE::SolicitTimer, _IP6_NEIGHBOR_ENTRY::State, _IP6_NEIGHBOR_ENTRY::Ticks, _IP6_SERVICE::Ticks, _IP6_NEIGHBOR_ENTRY::Transmit, and _IP6_DAD_ENTRY::Transmit.

Referenced by Ip6CreateService().

```
VOID Ip6NdTimerTicking ( IN IP6_SERVICE *  IpSb  )
```

The heartbeat timer of ND module in 1 second. This time routine handles following things: 1) maitain default router list; 2) maintain prefix options; 3) maintain route caches.

Parameters:

[in] *IpSb* The IP6 service binding instance.

References IP6_INF_ROUTER_LIFETIME, IP6_INFINIT_LIFETIME, IP6_ROUTE_CACHE_HASH_SIZE, IP6_ROUTE_CACHE_MAX, Ip6DestroyDefaultRouter(), Ip6DestroyPrefixListEntry(), Ip6FreeRouteCacheEntry(), _IP6_DEFAULT_ROUTER::Lifetime, _IP6_PREFIX_LIST_ENTRY::PreferredLifetime, and _IP6_PREFIX_LIST_ENTRY::ValidLifetime.

Referenced by Ip6TimerTicking().

VOID Ip6OnArpResolved (IN VOID * *Context*)

Callback function when address resolution is finished. It will cancel all the queued frames if the address resolution failed, or transmit them if the request succeeded.

Parameters:

[in] *Context* The context of the callback, a pointer to IP6_NEIGHBOR_ENTRY.

References IP6_LINK_TX_TOKEN::CallBack, IP6_LINK_TX_TOKEN::Context, IP6_LINK_TX_TOKEN::DstMac, _IP6_NEIGHBOR_ENTRY::Frames, _IP6_NEIGHBOR_ENTRY::Interface, IP6_DELAY_FIRST_PROBE_TIME, IP6_GET_TICKS, IP6_SERVICE_SIGNATURE, Ip6FreeLinkTxToken(), Ip6FreeNeighborEntry(), IP6_LINK_TX_TOKEN::Link, _IP6_NEIGHBOR_ENTRY::LinkAddress, _IP6_SERVICE::Mnp, IP6_LINK_TX_TOKEN::MnpToken, mZeroMacAddress, IP6_LINK_TX_TOKEN::Packet, _IP6_INTERFACE::SentFrames, _IP6_INTERFACE::Service, _IP6_SERVICE::Signature, _IP6_SERVICE::SnpMode, _IP6_NEIGHBOR_ENTRY::State, and _IP6_NEIGHBOR_ENTRY::Ticks.

Referenced by Ip6AddNeighbor(), Ip6Output(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), Ip6ProcessRouterAdvertise(), and Ip6SendNeighborSolicit().

VOID Ip6OnDADFinished (IN BOOLEAN *IsDadPassed*,
IN IP6_INTERFACE * *IpIf*,
IN IP6_DAD_ENTRY * *DadEntry*
)

The function to be called after DAD (Duplicate Address Detection) is performed.

Parameters:

[in] *IsDadPassed* If TRUE, the DAD operation succeed. Otherwise, the DAD operation failed.
[in] *IpIf* Points to the IP6_INTERFACE.
[in] *DadEntry* The DAD entry which already performed DAD.

References _IP6_ADDRESS_INFO::Address, _IP6_CONFIG_INSTANCE::Dhcp6, _IP6_CONFIG_INSTANCE::Dhcp6Event, _IP6_SERVICE::Dhcp6NeedInfoRequest, _IP6_SERVICE::Dhcp6NeedStart, _IP6_SERVICE::FasterTimer, IP6_CONFIG_DHCP6_OPTION_DNS_SERVERS, IP6_CONFIG_DHCP6_OPTION_ORO, IP6_INFINIT_LIFETIME, IP6_LINK_LOCAL_PREFIX_LENGTH, IP6_LINK_LOCAL_SCOPE, Ip6AddAddr(), _IP6_SERVICE::Ip6ConfigInstance, Ip6ConfigOnDhcp6Reply(), Ip6CreatePrefixListEntry(), Ip6LeaveGroup(), Ip6SetToAllNodeMulticast(), _IP6_SERVICE::LinkLocalAddr, _IP6_SERVICE::LinkLocalDadFail, _IP6_SERVICE::LinkLocalOk, _IP6_SERVICE::Mnp,

_IP6_ADDRESS_INFO::PreferredLifetime, _IP6_ADDRESS_INFO::PrefixLength, _IP6_SERVICE::Timer, and _IP6_ADDRESS_INFO::ValidLifetime.

Referenced by Ip6ConfigOnPolicyChanged(), Ip6InitDADProcess(), Ip6NdFasterTimerTicking(), Ip6ProcessNeighborAdvertise(), and Ip6ProcessNeighborSolicit().

```
EFI_STATUS Ip6ProcessNeighborAdvertise ( IN IP6_SERVICE *        IpSb,
                                         IN EFI_IP6_HEADER *     Head,
                                         IN NET_BUF *            Packet
                                       )
```

Process the Neighbor Advertisement message.

Parameters:

[in]	*IpSb*	The IP service that received the packet.
[in]	*Head*	The IP head of the message.
[in]	*Packet*	The content of the message with IP head removed.

Return values:

EFI_SUCCESS	The packet processed successfully.
EFI_INVALID_PARAMETER	The packet is invalid.
EFI_ICMP_ERROR	The packet indicates that DAD is failed.
Others	Failed to process the packet.

References _IP6_NEIGHBOR_ENTRY::CallBack, _IP6_ETHE_ADDR_OPTION::EtherAddr, IP6_GET_TICKS, IP6_HOP_LIMIT, IP6_INFINIT_LIFETIME, IP6_IS_ROUTER_FLAG, IP6_ND_LENGTH, IP6_OVERRIDE_FLAG, IP6_SOLICITED_FLAG, Ip6DestroyDefaultRouter(), Ip6FindDADEntry(), Ip6FindDefaultRouter(), Ip6FindNeighborEntry(), Ip6IsNDOptionValid(), Ip6OnDADFinished(), Ip6OptionEtherTarget, _IP6_NEIGHBOR_ENTRY::IsRouter, _IP6_NEIGHBOR_ENTRY::LinkAddress, _IP6_DEFAULT_ROUTER::NeighborCache, _IP6_NEIGHBOR_ENTRY::State, and _IP6_NEIGHBOR_ENTRY::Ticks.

Referenced by Ip6ProcessIcmpInformation().

```
EFI_STATUS Ip6ProcessNeighborSolicit ( IN IP6_SERVICE *        IpSb,
                                       IN EFI_IP6_HEADER *     Head,
                                       IN NET_BUF *            Packet
                                     )
```

Process the Neighbor Solicitation message. The message may be sent for Duplicate Address Detection or Address Resolution.

Parameters:

[in]	*IpSb*	The IP service that received the packet.
[in]	*Head*	The IP head of the message.
[in]	*Packet*	The content of the message with IP head removed.

Return values:

EFI_SUCCESS	The packet processed successfully.
EFI_INVALID_PARAMETER	The packet is invalid.
EFI_ICMP_ERROR	The packet indicates that DAD is failed.
Others	Failed to process the packet.

References _IP6_NEIGHBOR_ENTRY::CallBack, _IP6_ETHE_ADDR_OPTION::EtherAddr, IP6_HOP_LIMIT, IP6_INFINIT_LIFETIME, IP6_LINK_LOCAL_SCOPE, IP6_ND_LENGTH, Ip6CreateNeighborEntry(), Ip6FindDADEntry(), Ip6FindNeighborEntry(), Ip6IsNDOptionValid(), Ip6IsOneOfSetAddress(), Ip6IsSNMulticastAddr(), Ip6OnArpResolved(), Ip6OnDADFinished(), Ip6OptionEtherSource, Ip6SendNeighborAdvertise(), Ip6SetToAllNodeMulticast(), _IP6_NEIGHBOR_ENTRY::LinkAddress, _IP6_DAD_ENTRY::Receive, _IP6_NEIGHBOR_ENTRY::State, and _IP6_NEIGHBOR_ENTRY::Ticks.

Referenced by Ip6ProcessIcmpInformation().

```
EFI_STATUS Ip6ProcessRedirect ( IN IP6_SERVICE *        IpSb,
                                IN EFI_IP6_HEADER *     Head,
                                IN NET_BUF *            Packet
                              )
```

Process the ICMPv6 redirect message. Find the instance, then update its route cache.

Parameters:

[in]	*IpSb*	The IP6 service binding instance that received the packet.
[in]	*Head*	The IP head of the received ICMPv6 packet.
[in]	*Packet*	The content of the ICMPv6 redirect packet with the IP head removed.

Return values:

EFI_INVALID_PARAMETER	The parameter is invalid.
EFI_OUT_OF_RESOURCES	Insuffcient resources to complete the operation.
EFI_SUCCESS	Successfully updated the route caches.

References _IP6_NEIGHBOR_ENTRY::CallBack, _IP6_ETHE_ADDR_OPTION::EtherAddr, IP6_ROUTE_ENTRY::Flag, IP6_DIRECT_ROUTE, IP6_HOP_LIMIT, IP6_INFINIT_LIFETIME, IP6_REDITECT_LENGTH, IP6_ROUTE_CACHE_HASH, Ip6CreateNeighborEntry(), Ip6CreateRouteCacheEntry(), Ip6CreateRouteEntry(), Ip6FindNeighborEntry(), Ip6FindRouteCache(), Ip6FindRouteEntry(), Ip6IsNDOptionValid(), Ip6IsOneOfSetAddress(), Ip6OnArpResolved(), Ip6OptionEtherTarget, _IP6_NEIGHBOR_ENTRY::IsRouter, _IP6_ETHE_ADDR_OPTION::Length, IP6_ROUTE_CACHE_ENTRY::Link, _IP6_NEIGHBOR_ENTRY::LinkAddress, IP6_ROUTE_CACHE_ENTRY::NextHop, _IP6_NEIGHBOR_ENTRY::State, IP6_ROUTE_CACHE_ENTRY::Tag, and _IP6_NEIGHBOR_ENTRY::Ticks.

Referenced by Ip6ProcessIcmpInformation().

```
EFI_STATUS Ip6ProcessRouterAdvertise ( IN IP6_SERVICE *        IpSb,
                                       IN EFI_IP6_HEADER *     Head,
                                       IN NET_BUF *            Packet
                                     )
```

Process the Router Advertisement message according to RFC4861.

Parameters:

[in]	*IpSb*	The IP service that received the packet.
[in]	*Head*	The IP head of the message.
[in]	*Packet*	The content of the message with the IP head removed.

Return values:

EFI_SUCCESS	The packet processed successfully.
EFI_INVALID_PARAMETER	The packet is invalid.
EFI_OUT_OF_RESOURCES	Insufficient resources to complete the operation.
Others	Failed to process the packet.

Process the Router Advertisement message according to RFC4861.

Parameters:

[in]	*IpSb*	The IP service that received the packet.
[in]	*Head*	The IP head of the message.
[in]	*Packet*	The content of the message with the IP head removed.

Return values:

EFI_SUCCESS	The packet processed successfully.
EFI_INVALID_PARAMETER	The packet is invalid.
EFI_OUT_OF_RESOURCES	Insufficient resources to complete the operation.
Others	Failed to process the packet.

References _IP6_NEIGHBOR_ENTRY::CallBack, IP6_AUTO_CONFIG_FLAG, IP6_HOP_LIMIT, IP6_INFINIT_LIFETIME, IP6_LINK_LOCAL_PREFIX_LENGTH, IP6_M_ADDR_CONFIG_FLAG, IP6_O_CONFIG_FLAG, IP6_ON_LINK_FLAG, IP6_RA_LENGTH, Ip6ConfigStartStatefulAutoConfig(), Ip6CreateDefaultRouter(), Ip6CreateNeighborEntry(), Ip6CreatePrefixListEntry(), Ip6DestroyDefaultRouter(), Ip6DestroyPrefixListEntry(), Ip6FindDADEntry(), Ip6FindDefaultRouter(), Ip6FindNeighborEntry(), Ip6FindPrefixListEntry(), Ip6IsNDOptionValid(), Ip6IsOneOfSetAddress(), Ip6OnArpResolved(), Ip6OptionEtherSource, Ip6OptionMtu, Ip6OptionPrefixInfo, Ip6SetAddress(), Ip6UpdateReachableTime(), _IP6_NEIGHBOR_ENTRY::IsRouter, _IP6_DEFAULT_ROUTER::Lifetime, _IP6_NEIGHBOR_ENTRY::LinkAddress, _IP6_DEFAULT_ROUTER::NeighborCache, _IP6_PREFIX_LIST_ENTRY::PreferredLifetime, _IP6_NEIGHBOR_ENTRY::State, _IP6_NEIGHBOR_ENTRY::Ticks, and _IP6_PREFIX_LIST_ENTRY::ValidLifetime.

Referenced by Ip6ProcessIcmpInformation().

```
EFI_STATUS Ip6SendNeighborSolicit ( IN IP6_SERVICE *          IpSb,
                                    IN EFI_IPv6_ADDRESS *     SourceAddress,
                                    IN EFI_IPv6_ADDRESS *     DestinationAddress,
                                    IN EFI_IPv6_ADDRESS *     TargetIp6Address,
                                    IN EFI_MAC_ADDRESS *      SourceLinkAddress
                                  )
```

Generate the Neighbor Solicitation message and send it to the Destination Address.

Parameters:

[in]	*IpSb*	The IP service to send the packet
[in]	*SourceAddress*	The source address of the message.
[in]	*DestinationAddress*	The destination address of the message.
[in]	*TargetIp6Address*	The IP address of the target of the solicitation. It must not be a multicast address.
[in]	*SourceLinkAddress*	The MAC address for the sender. If not NULL, a source link-layer address option will be appended to the message.

Return values:

EFI_INVALID_PARAMETER	Any input parameter is invalid.

EFI_OUT_OF_RESOURCES	Insufficient resources to complete the operation.
EFI_SUCCESS	The Neighbor Advertise message was successfully sent.

References _IP6_ETHE_ADDR_OPTION::EtherAddr, IP6_HOP_LIMIT, IP6_SERVICE_SIGNATURE, Ip6CreateNeighborEntry(), Ip6FindNeighborEntry(), Ip6IsSNMulticastAddr(), Ip6OnArpResolved(), Ip6OptionEtherSource, Ip6Output(), Ip6SysPacketSent(), _IP6_ETHE_ADDR_OPTION::Length, and _IP6_ETHE_ADDR_OPTION::Type.

Referenced by Ip6NdFasterTimerTicking(), and Ip6Output().

```
EFI_STATUS Ip6SendRouterSolicit ( IN IP6_SERVICE *          IpSb,
                                  IN IP6_INTERFACE *        Interface,
                                  IN EFI_IPv6_ADDRESS *     SourceAddress,
                                  IN EFI_IPv6_ADDRESS *     DestinationAddress,
                                  IN EFI_MAC_ADDRESS *      SourceLinkAddress
                                )
```

Generate router solicit message and send it out to Destination Address or All Router Link Local scope multicast address.

Parameters:

[in]	*IpSb*	The IP service to send the packet.
[in]	*Interface*	If not NULL, points to the IP6 interface to send the packet.
[in]	*SourceAddress*	If not NULL, the source address of the message.
[in]	*DestinationAddress*	If not NULL, the destination address of the message.
[in]	*SourceLinkAddress*	If not NULL, the MAC address of the source. A source link-layer address option will be appended to the message.

Return values:

EFI_OUT_OF_RESOURCES	Insufficient resources to complete the operation.
EFI_SUCCESS	The router solicit message was successfully sent.

Generate router solicit message and send it out to Destination Address or All Router Link Local scope multicast address.

Parameters:

[in]	*IpSb*	The IP service to send the packet.
[in]	*Interface*	If not NULL, points to the IP6 interface to send the packet.
[in]	*SourceAddress*	If not NULL, the source address of the message.
[in]	*DestinationAddress*	If not NULL, the destination address of the message.
[in]	*SourceLinkAddress*	If not NULL, the MAC address of the source. A source link-layer address option will be appended to the message.

Return values:

EFI_OUT_OF_RESOURCES	Insufficient resources to complete the operation.
EFI_SUCCESS	The router solicit message was successfully sent.

References _IP6_ETHE_ADDR_OPTION::EtherAddr, IP6_HOP_LIMIT, IP6_LINK_LOCAL_SCOPE, IP6_SERVICE_SIGNATURE, Ip6OptionEtherSource, Ip6Output(), Ip6SetToAllNodeMulticast(), Ip6SysPacketSent(), _IP6_ETHE_ADDR_OPTION::Length, and _IP6_ETHE_ADDR_OPTION::Type.

Referenced by Ip6NdFasterTimerTicking().

```
EFI_STATUS Ip6SetAddress ( IN IP6_INTERFACE *        Interface,
                           IN EFI_IPv6_ADDRESS *     Ip6Addr,
                           IN BOOLEAN                IsAnycast,
                           IN UINT8                  PrefixLength,
                           IN UINT32                 ValidLifetime,
                           IN UINT32                 PreferredLifetime,
                           IN IP6_DAD_CALLBACK       DadCallback,
                           IN VOID *                 Context
                         )
```

Set the interface's address. This will trigger the DAD process for the address to set. To set an already set address, the lifetimes wil be updated to the new value passed in.

Parameters:

[in]	*Interface*	The interface to set the address.
[in]	*Ip6Addr*	The interface's to be assigned IPv6 address.
[in]	*IsAnycast*	If TRUE, the unicast IPv6 address is anycast. Otherwise, it is not anycast.
[in]	*PrefixLength*	The prefix length of the Ip6Addr.
[in]	*ValidLifetime*	The valid lifetime for this address.
[in]	*PreferredLifetime*	The preferred lifetime for this address.
[in]	*DadCallback*	The caller's callback to trigger when DAD finishes. This is an optional parameter that may be NULL.
[in]	*Context*	The context that will be passed to DadCallback. This is an optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The interface is scheduled to be configured with the specified address.
EFI_OUT_OF_RESOURCES	Failed to set the interface's address due to lack of resources.

References _IP6_ADDRESS_INFO::Address, _IP6_DELAY_JOIN_LIST::AddressInfo, _IP6_SERVICE::AutonomousPrefix, _IP6_DELAY_JOIN_LIST::Context, _IP6_DELAY_JOIN_LIST::DadCallback, _IP6_DELAY_JOIN_LIST::DelayTime, _IP6_DELAY_JOIN_LIST::Interface, IP6_ADDR_INFO_SIGNATURE, IP6_INTERFACE_SIGNATURE, IP6_LINK_LOCAL_PREFIX_LENGTH, IP6_ONE_SECOND_IN_MS, IP6_TIMER_INTERVAL_IN_MS, Ip6IsOneOfSetAddress(), _IP6_ADDRESS_INFO::IsAnycast, _IP6_DELAY_JOIN_LIST::Link, _IP6_SERVICE::OnlinkPrefix, _IP6_ADDRESS_INFO::PreferredLifetime, _IP6_PREFIX_LIST_ENTRY::Prefix, _IP6_PREFIX_LIST_ENTRY::PrefixLength, _IP6_ADDRESS_INFO::PrefixLength, _IP6_ADDRESS_INFO::Signature, and _IP6_ADDRESS_INFO::ValidLifetime.

Referenced by Ip6ConfigOnDhcp6Event(), Ip6ConfigSetMaunualAddress(), Ip6CreateInterface(), and Ip6ProcessRouterAdvertise().

```
VOID Ip6UpdateReachableTime ( IN OUT IP6_SERVICE *   IpSb   )
```

Update the ReachableTime in IP6 service binding instance data, in milliseconds.

Parameters:

[in,out]	*IpSb*	Points to the IP6_SERVICE.

References IP6_MIN_RANDOM_FACTOR_SCALED, and IP6_RANDOM_FACTOR_SCALE.

Referenced by Ip6CreateService(), and Ip6ProcessRouterAdvertise().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Ip6Dxe/Ip6NvData.h File Reference

Data Structures

struct IP6_CONFIG_IFR_NVDATA

Defines

"memItemRight" valign="bottom">FORMID_MAIN_FORM 1

#define FORMID_MANUAL_CONFIG_FORM 2
#define FORMID_HEAD_FORM 3
#define IP6_POLICY_AUTO 0
#define IP6_POLICY_MANUAL 1
#define DAD_MAX_TRANSMIT_COUNT 10
#define KEY_INTERFACE_ID 0x101
#define KEY_MANUAL_ADDRESS 0x102
#define KEY_GATEWAY_ADDRESS 0x103
#define KEY_DNS_ADDRESS 0x104
#define KEY_SAVE_CHANGES 0x105
#define KEY_SAVE_CONFIG_CHANGES 0x106
#define KEY_IGNORE_CONFIG_CHANGES 0x107
#define KEY_GET_CURRENT_SETTING 0x108
#define HOST_ADDRESS_LABEL 0x9000
#define ROUTE_TABLE_LABEL 0xa000
#define GATEWAY_ADDRESS_LABEL 0xb000
#define DNS_ADDRESS_LABEL 0xc000
#define LABEL_END 0xffff
#define INTERFACE_ID_STR_MIN_SIZE 1
#define INTERFACE_ID_STR_MAX_SIZE 23
#define INTERFACE_ID_STR_STORAGE 25
#define IP6_STR_MAX_SIZE 40
#define ADDRESS_STR_MIN_SIZE 2
#define ADDRESS_STR_MAX_SIZE 255

Detailed Description

NVData structure used by the IP6 configuration component.

Copyright (c) 2010 - 2013, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Ip6ConvertInterfaceInfoToString(), and Ip6GetCurrentSetting().

#define ADDRESS_STR_MIN_SIZE 2

#define DAD_MAX_TRANSMIT_COUNT 10

#define DNS_ADDRESS_LABEL 0xc000

Referenced by Ip6ConvertAddressListToString().

#define FORMID_HEAD_FORM 3

#define FORMID_MAIN_FORM 1

Referenced by Ip6ConvertAddressListToString(), and IScsiConfigUpdateAttempt().

#define FORMID_MANUAL_CONFIG_FORM 2

#define GATEWAY_ADDRESS_LABEL 0xb000

Referenced by Ip6ConvertAddressListToString().

#define HOST_ADDRESS_LABEL 0x9000

Referenced by Ip6ConvertAddressListToString().

#define INTERFACE_ID_STR_MAX_SIZE 23

#define INTERFACE_ID_STR_MIN_SIZE 1

#define INTERFACE_ID_STR_STORAGE 25

Referenced by Ip6ConvertInterfaceIdToString().

#define IP6_POLICY_AUTO 0

Referenced by Ip6ConvertConfigNvDataToIfrNvData(), Ip6ConvertIfrNvDataToConfigNvDataAdvanced(), and Ip6ConvertIfrNvDataToConfigNvDataGeneral().

#define IP6_POLICY_MANUAL 1

Referenced by Ip6ConvertConfigNvDataToIfrNvData(), and Ip6ConvertIfrNvDataToConfigNvDataGeneral().

#define IP6_STR_MAX_SIZE 40

Referenced by Ip6ToStr().

#define KEY_DNS_ADDRESS 0x104

Referenced by Ip6FormCallback().

#define KEY_GATEWAY_ADDRESS 0x103

Referenced by Ip6FormCallback().

#define KEY_GET_CURRENT_SETTING 0x108

Referenced by Ip6FormCallback().

#define KEY_IGNORE_CONFIG_CHANGES 0x107

Referenced by Ip6FormCallback().

#define KEY_INTERFACE_ID 0x101

Referenced by Ip6FormCallback().

#define KEY_MANUAL_ADDRESS 0x102

Referenced by Ip6FormCallback().

#define KEY_SAVE_CHANGES 0x105

Referenced by Ip6FormCallback().

#define KEY_SAVE_CONFIG_CHANGES 0x106

Referenced by Ip6FormCallback().

label LABEL_END 0xffff

Referenced by Ip6CreateOpCode(), and IScsiCreateOpCode().

#define ROUTE_TABLE_LABEL 0xa000

Referenced by Ip6ConvertAddressListToString().

NetworkPkg/Ip6Dxe/Ip6Option.c File Reference

Functions

BOOLEAN Ip6IsOptionValid (IN IP6_SERVICE *IpSb, IN NET_BUF *Packet, IN UINT8 *Option, IN UINT8 OptionLen, IN UINT32 Pointer)
BOOLEAN Ip6IsNDOptionValid (IN UINT8 *Option, IN UINT16 OptionLen)
BOOLEAN Ip6IsValidProtocol (IN IP6_SERVICE *IpSb, IN UINT8 NextHeader)
BOOLEAN Ip6IsExtsValid (IN IP6_SERVICE *IpSb, IN NET_BUF *Packet, IN UINT8 *NextHeader, IN UINT8 *ExtHdrs, IN UINT32 ExtHdrsLen, IN BOOLEAN Rcvd, OUT UINT32 *FormerHeader, OUT UINT8 **LastHeader, OUT UINT32 *RealExtsLen, OUT UINT32 *UnFragmentLen, OUT BOOLEAN *Fragmented)
EFI_STATUS Ip6FillHopByHop (OUT UINT8 *Buffer, IN OUT UINTN *BufferLen, IN UINT8 NextHeader)
EFI_STATUS Ip6FillFragmentHeader (IN IP6_SERVICE *IpSb, IN UINT8 NextHeader, IN UINT8 LastHeader, IN UINT8 *ExtHdrs, IN UINT32 ExtHdrsLen, IN UINT16 FragmentOffset, OUT UINT8 **UpdatedExtHdrs)

Detailed Description

IP6 option support functions and routines.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS Ip6FillFragmentHeader ( IN IP6_SERVICE *   IpSb,
                                   IN UINT8           NextHeader,
                                   IN UINT8           LastHeader,
                                   IN UINT8 *         ExtHdrs,
                                   IN UINT32          ExtHdrsLen,
                                   IN UINT16          FragmentOffset,
                                   OUT UINT8 **       UpdatedExtHdrs
                                 )
```

Insert a Fragment Header to the Extension headers and output it in UpdatedExtHdrs.

Parameters:

[in]	*IpSb*	The IP6 service instance to transmit the packet.
[in]	*NextHeader*	The extension header type of first extension header.
[in]	*LastHeader*	The extension header type of last extension header.

[in]	*ExtHdrs*	The length of the original extension header.
[in]	*ExtHdrsLen*	The length of the extension headers.
[in]	*FragmentOffset*	The fragment offset of the data following the header.
[out]	*UpdatedExtHdrs*	The updated ExtHdrs with Fragment header inserted. It's caller's responsiblity to free this buffer.

Return values:

EFI_OUT_OF_RESOURCES	Failed to finish the operation due to lake of resource.
EFI_UNSUPPORTED	The extension header specified in ExtHdrs is not supported currently.
EFI_SUCCESS	The operation performed successfully.

References _IP6_FRAGMENT_HEADER::FragmentOffset, _IP6_FRAGMENT_HEADER::Identification, IP6_FRAGMENT_OFFSET_MASK, Ip6IsValidProtocol(), mIp6Id, _IP6_FRAGMENT_HEADER::NextHeader, and _IP6_FRAGMENT_HEADER::Reserved.

Referenced by Ip6Output(), and Ip6PrependHead().

```
EFI_STATUS Ip6FillHopByHop (  OUT UINT8 *      Buffer,
                              IN OUT UINTN *   BufferLen,
                              IN UINT8         NextHeader
                            )
```

Generate an IPv6 router alert option in network order and output it through Buffer.

Parameters:

[out]	*Buffer*	Points to a buffer to record the generated option.
[in,out]	*BufferLen*	The length of Buffer, in bytes.
[in]	*NextHeader*	The 8-bit selector indicates the type of header immediately following the Hop-by-Hop Options header.

Return values:

EFI_BUFFER_TOO_SMALL	The Buffer is too small to contain the generated option. BufferLen is updated for the required size.
EFI_SUCCESS	The option is generated and filled in to Buffer.

Referenced by Ip6SendMldDone(), and Ip6SendMldReport().

```
BOOLEAN Ip6IsExtsValid ( IN IP6_SERVICE *   IpSb,
                         IN NET_BUF *       Packet,
                         IN UINT8 *         NextHeader,
                         IN UINT8 *         ExtHdrs,
                         IN UINT32          ExtHdrsLen,
                         IN BOOLEAN         Rcvd,
                         OUT UINT32 *       FormerHeader,
                         OUT UINT8 **       LastHeader,
                         OUT UINT32 *       RealExtsLen,
                         OUT UINT32 *       UnFragmentLen,
```

OUT BOOLEAN * *Fragmented*
)

Validate the IP6 extension header format for both the packets we received and that we will transmit. It will compute the ICMPv6 error message fields if the option is mal-formated.

Parameters:

[in]	*IpSb*	The IP6 service instance. This is an optional parameter.
[in]	*Packet*	The data of the packet. Ignored if NULL.
[in]	*NextHeader*	The next header field in IPv6 basic header.
[in]	*ExtHdrs*	The first byte of the option.
[in]	*ExtHdrsLen*	The length of the whole option.
[in]	*Rcvd*	The option is from the packet we received if TRUE, otherwise, the option we want to transmit.
[out]	*FormerHeader*	The offset of NextHeader which points to Fragment Header when we received, of the ExtHdrs. Ignored if we transmit.
[out]	*LastHeader*	The pointer of NextHeader of the last extension header processed by IP6.
[out]	*RealExtsLen*	The length of extension headers processed by IP6 layer. This is an optional parameter that may be NULL.
[out]	*UnFragmentLen*	The length of unfragmented length of extension headers. This is an optional parameter that may be NULL.
[out]	*Fragmented*	Indicate whether the packet is fragmented. This is an optional parameter that may be NULL.

Return values:

TRUE The option is properly formated.
FALSE The option is malformated.

References _IP6_FRAGMENT_HEADER::FragmentOffset, _IP6_ROUTING_HEADER::HeaderLen, Ip6IsOptionValid(), Ip6IsValidProtocol(), Ip6SendIcmpError(), and _IP6_ROUTING_HEADER::SegmentsLeft.

Referenced by Ip6InstanceFrameAcceptable(), Ip6Output(), and Ip6PreProcessPacket().

BOOLEAN Ip6IsNDOptionValid (IN UINT8 * *Option*,
IN UINT16 *OptionLen*
)

Validate the IP6 option format for both the packets we received and that we will transmit. It supports the defined options in Neighbor Discovery messages.

Parameters:

[in]	*Option*	The first byte of the option.
[in]	*OptionLen*	The length of the whole option.

Return values:

TRUE The option is properly formatted.
FALSE The option is malformated.

References Ip6OptionMtu, and Ip6OptionPrefixInfo.

Referenced by Ip6ProcessNeighborAdvertise(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), and Ip6ProcessRouterAdvertise().

```
BOOLEAN Ip6IsOptionValid ( IN IP6_SERVICE *   IpSb,
                           IN NET_BUF *       Packet,
                           IN UINT8 *         Option,
                           IN UINT8           OptionLen,
                           IN UINT32          Pointer
                         )
```

Validate the IP6 option format for both the packets we received and that we will transmit. It will compute the ICMPv6 error message fields if the option is malformated.

Parameters:

[in]	*IpSb*	The IP6 service data.
[in]	*Packet*	The to be validated packet.
[in]	*Option*	The first byte of the option.
[in]	*OptionLen*	The length of the whole option.
[in]	*Pointer*	Identifies the octet offset within the invoking packet where the error was detected.

Return values:

TRUE	The option is properly formatted.
FALSE	The option is malformated.

References Ip6OptionDiscard, Ip6OptionMask, Ip6OptionPad1, Ip6OptionPadN, Ip6OptionParameterProblem, Ip6OptionRouterAlert, Ip6OptionSkip, and Ip6SendIcmpError().

Referenced by Ip6IsExtsValid().

```
BOOLEAN Ip6IsValidProtocol ( IN IP6_SERVICE *   IpSb,
                             IN UINT8           NextHeader
                           )
```

Validate whether the NextHeader is a known valid protocol or one of the user configured protocols from the upper layer.

Parameters:

[in]	*IpSb*	The IP6 service instance.
[in]	*NextHeader*	The next header field.

Return values:

TRUE	The NextHeader is a known valid protocol or user configured.
FALSE	The NextHeader is not a known valid protocol.

References _IP6_PROTOCOL::ConfigData, IP6_PROTOCOL_SIGNATURE, IP6_SERVICE_SIGNATURE, IP6_STATE_CONFIGED, and _IP6_PROTOCOL::State.

Referenced by Ip6FillFragmentHeader(), and Ip6IsExtsValid().

- Globals

NetworkPkg/Ip6Dxe/Ip6Option.h File Reference

Data Structures

struct _IP6_FRAGMENT_HEADER
struct _IP6_ROUTING_HEADER

Defines

"memItemRight" valign="bottom">IP6_FRAGMENT_OFFSET_MASK (~0x3)

Typedefs

typedef struct _IP6_FRAGMENT_HEADER IP6_FRAGMENT_HEADER
typedef struct _IP6_ROUTING_HEADER IP6_ROUTING_HEADER

Enumerations

enum IP6_OPTION_TYPE {
Ip6OptionPad1 = 0,
Ip6OptionPadN = 1,
Ip6OptionRouterAlert = 5,
Ip6OptionSkip = 0,
Ip6OptionDiscard = 0x40,
Ip6OptionParameterProblem = 0x80, Ip6OptionMask = 0xc0,
Ip6OptionEtherSource = 1,
Ip6OptionEtherTarget = 2,
Ip6OptionPrefixInfo = 3,
Ip6OptionRedirected = 4,
Ip6OptionMtu = 5
}

Functions

BOOLEAN Ip6IsExtsValid (IN IP6_SERVICE *IpSb, IN NET_BUF *Packet, IN UINT8 *NextHeader, IN UINT8 *ExtHdrs, IN UINT32 ExtHdrsLen, IN BOOLEAN Rcvd, OUT UINT32 *FormerHeader, OUT UINT8 **LastHeader, OUT UINT32 *RealExtsLen, OUT UINT32 *UnFragmentLen, OUT BOOLEAN *Fragmented)

EFI_STATUS Ip6FillHopByHop (OUT UINT8 *Buffer, IN OUT UINTN *BufferLen, IN

UINT8 NextHeader)
EFI_STATUS Ip6FillFragmentHeader (IN IP6_SERVICE *IpSb, IN UINT8 NextHeader, IN UINT8 LastHeader, IN UINT8 *ExtHdrs, IN UINT32 ExtHdrsLen, IN UINT16 FragmentOffset, OUT UINT8 **UpdatedExtHdrs)
EFI_STATUS Ip6CopyExts (IN UINT8 NextHeader, IN UINT8 *ExtHdrs, IN UINT8 *LastHeader, IN UINT16 FragmentOffset, IN UINT32 UnFragmentHdrLen, IN OUT UINT8 *Buf, IN OUT UINT32 *BufLen)
BOOLEAN Ip6IsNDOptionValid (IN UINT8 *Option, IN UINT16 OptionLen)

Detailed Description

Definition of IP6 option process routines.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Ip6FillFragmentHeader(), and Ip6PrependHead().

Typedef Documentation

typedef struct _IP6_FRAGMENT_HEADER IP6_FRAGMENT_HEADER

typedef struct _IP6_ROUTING_HEADER IP6_ROUTING_HEADER

Enumeration Type Documentation

enum IP6_OPTION_TYPE

Enumerator:

Ip6OptionPad1
Ip6OptionPadN
Ip6OptionRouterAlert
Ip6OptionSkip
Ip6OptionDiscard
Ip6OptionParameterProblem
Ip6OptionMask
Ip6OptionEtherSource
Ip6OptionEtherTarget
Ip6OptionPrefixInfo
Ip6OptionRedirected
Ip6OptionMtu

Function Documentation

```
EFI_STATUS Ip6CopyExts ( IN UINT8          NextHeader,
                         IN UINT8 *        ExtHdrs,
                         IN UINT8 *        LastHeader,
                         IN UINT16         FragmentOffset,
                         IN UINT32         UnFragmentHdrLen,
                         IN OUT UINT8 *    Buf,
                         IN OUT UINT32 *   BufLen
                       )
```

Copy the extension headers from the original to buffer. A Fragment header is appended to the end.

Parameters:

[in]	*NextHeader*	The 8-bit selector indicates the type of the fragment header's next header.
[in]	*ExtHdrs*	The length of the original extension header.
[in]	*LastHeader*	The pointer of next header of last extension header.
[in]	*FragmentOffset*	The fragment offset of the data following the header.
[in]	*UnFragmentHdrLen*	The length of unfragmented length of extension headers.
[in,out]	*Buf*	The buffer to copy options to.
[in,out]	*BufLen*	The length of the buffer.

Return values:

EFI_SUCCESS	The options are copied over.
EFI_BUFFER_TOO_SMALL	The buffer caller provided is too small.

```
EFI_STATUS Ip6FillFragmentHeader ( IN IP6_SERVICE *  IpSb,
                                   IN UINT8          NextHeader,
```

```
                    IN UINT8          LastHeader,
                    IN UINT8 *        ExtHdrs,
                    IN UINT32         ExtHdrsLen,
                    IN UINT16         FragmentOffset,
                    OUT UINT8 **      UpdatedExtHdrs
                  )
```

Insert a Fragment Header to the Extension headers and output it in UpdatedExtHdrs.

Parameters:

[in]	*IpSb*	The IP6 service instance to transmit the packet.
[in]	*NextHeader*	The extension header type of first extension header.
[in]	*LastHeader*	The extension header type of last extension header.
[in]	*ExtHdrs*	The length of the original extension header.
[in]	*ExtHdrsLen*	The length of the extension headers.
[in]	*FragmentOffset*	The fragment offset of the data following the header.
[out]	*UpdatedExtHdrs*	The updated ExtHdrs with Fragment header inserted. It's caller's responsiblity to free this buffer.

Return values:

EFI_OUT_OF_RESOURCES	Failed to finish the operation due to lake of resource.
EFI_UNSUPPORTED	The extension header specified in ExtHdrs is not supported currently.
EFI_SUCCESS	The operation performed successfully.

References _IP6_FRAGMENT_HEADER::FragmentOffset, _IP6_FRAGMENT_HEADER::Identification, IP6_FRAGMENT_OFFSET_MASK, Ip6IsValidProtocol(), mIp6Id, _IP6_FRAGMENT_HEADER::NextHeader, and _IP6_FRAGMENT_HEADER::Reserved.

Referenced by Ip6Output(), and Ip6PrependHead().

```
EFI_STATUS Ip6FillHopByHop  (  OUT UINT8 *      Buffer,
                               IN OUT UINTN *   BufferLen,
                               IN UINT8         NextHeader
                            )
```

Generate an IPv6 router alert option in network order and output it through Buffer.

Parameters:

[out]	*Buffer*	Points to a buffer to record the generated option.
[in,out]	*BufferLen*	The length of Buffer, in bytes.
[in]	*NextHeader*	The 8-bit selector indicates the type of header immediately following the Hop-by-Hop Options header.

Return values:

EFI_BUFFER_TOO_SMALL	The Buffer is too small to contain the generated option. BufferLen is updated for the required size.
EFI_SUCCESS	The option is generated and filled in to Buffer.

Referenced by Ip6SendMldDone(), and Ip6SendMldReport().

```
BOOLEAN Ip6IsExtsValid ( IN IP6_SERVICE *   IpSb,
                         IN NET_BUF *       Packet,
                         IN UINT8 *         NextHeader,
                         IN UINT8 *         ExtHdrs,
                         IN UINT32          ExtHdrsLen,
                         IN BOOLEAN         Rcvd,
                         OUT UINT32 *       FormerHeader,
                         OUT UINT8 **       LastHeader,
                         OUT UINT32 *       RealExtsLen,
                         OUT UINT32 *       UnFragmentLen,
                         OUT BOOLEAN *      Fragmented
                       )
```

Validate the IP6 extension header format for both the packets we received and that we will transmit. It will compute the ICMPv6 error message fields if the option is mal-formated.

Parameters:

[in]	*IpSb*	The IP6 service instance. This is an optional parameter.
[in]	*Packet*	The data of the packet. Ignored if NULL.
[in]	*NextHeader*	The next header field in IPv6 basic header.
[in]	*ExtHdrs*	The first byte of the option.
[in]	*ExtHdrsLen*	The length of the whole option.
[in]	*Rcvd*	The option is from the packet we received if TRUE, otherwise, the option we want to transmit.
[out]	*FormerHeader*	The offset of NextHeader which points to Fragment Header when we received, of the ExtHdrs. Ignored if we transmit.
[out]	*LastHeader*	The pointer of NextHeader of the last extension header processed by IP6.
[out]	*RealExtsLen*	The length of extension headers processed by IP6 layer. This is an optional parameter that may be NULL.
[out]	*UnFragmentLen*	The length of unfragmented length of extension headers. This is an optional parameter that may be NULL.
[out]	*Fragmented*	Indicate whether the packet is fragmented. This is an optional parameter that may be NULL.

Return values:

TRUE	The option is properly formated.
FALSE	The option is malformated.

References _IP6_FRAGMENT_HEADER::FragmentOffset, _IP6_ROUTING_HEADER::HeaderLen, Ip6IsOptionValid(), Ip6IsValidProtocol(), Ip6SendIcmpError(), and _IP6_ROUTING_HEADER::SegmentsLeft.

Referenced by Ip6InstanceFrameAcceptable(), Ip6Output(), and Ip6PreProcessPacket().

```
BOOLEAN Ip6IsNDOptionValid ( IN UINT8 *   Option,
                             IN UINT16    OptionLen
                           )
```

Validate the IP6 option format for both the packets we received and that we will transmit. It supports the defined options in Neighbor Discovery messages.

Parameters:

[in]	*Option*	The first byte of the option.
[in]	*OptionLen*	The length of the whole option.

Return values:

TRUE	The option is properly formatted.
FALSE	The option is malformated.

References Ip6OptionMtu, and Ip6OptionPrefixInfo.

Referenced by Ip6ProcessNeighborAdvertise(), Ip6ProcessNeighborSolicit(), Ip6ProcessRedirect(), and Ip6ProcessRouterAdvertise().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Ip6Dxe/Ip6Output.c File Reference

Functions

EFI_STATUS Ip6CandidateSource (IN IP6_SERVICE *IpSb, OUT LIST_ENTRY *SourceList, OUT UINT32 *SourceCount)

UINT8 Ip6CommonPrefixLen (IN EFI_IPv6_ADDRESS *AddressA, IN EFI_IPv6_ADDRESS *AddressB)

EFI_STATUS Ip6SelectSourceAddress (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *Destination, OUT EFI_IPv6_ADDRESS *Source)

IP6_INTERFACE * Ip6SelectInterface (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *Destination, IN OUT EFI_IPv6_ADDRESS *Source)

VOID Ip6SysPacketSent (NET_BUF *Packet, EFI_STATUS IoStatus, UINT32 LinkFlag, VOID *Context)

EFI_STATUS Ip6PrependHead (IN IP6_SERVICE *IpSb, IN NET_BUF *Packet, IN EFI_IP6_HEADER *Head, IN UINT16 FragmentOffset, IN UINT8 *ExtHdrs, IN UINT32 ExtHdrsLen, IN UINT8 LastHeader, IN UINT32 HeadLen)

EFI_STATUS Ip6Output (IN IP6_SERVICE *IpSb, IN IP6_INTERFACE *Interface, IN IP6_PROTOCOL *IpInstance, IN NET_BUF *Packet, IN EFI_IP6_HEADER *Head, IN UINT8 *ExtHdrs, IN UINT32 ExtHdrsLen, IN IP6_FRAME_CALLBACK Callback, IN VOID *Context)

BOOLEAN Ip6CancelPacketFragments (IN IP6_LINK_TX_TOKEN *Frame, IN VOID *Context)

VOID Ip6CancelFrames (IN IP6_INTERFACE *Interface, IN EFI_STATUS IoStatus, IN IP6_FRAME_TO_CANCEL FrameToCancel, IN VOID *Context)

VOID Ip6CancelPacket (IN IP6_INTERFACE *IpIf, IN NET_BUF *Packet, IN EFI_STATUS IoStatus)

Variables

UINT32 mIp6Id

Detailed Description

The internal functions and routines to transmit the IP6 packet.

Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

VOID Ip6CancelFrames (IN IP6_INTERFACE * *Interface*,
IN EFI_STATUS *IoStatus*,

```
                    IN IP6_FRAME_TO_CANCEL   FrameToCancel,
                    IN VOID *                Context
                )
```

Remove all the frames on the interface that pass the FrameToCancel, either queued on ARP queues or that have already been delivered to MNP and not yet recycled.

Parameters:

[in]	*Interface*	Interface to remove the frames from.
[in]	*IoStatus*	The transmit status returned to the frames' callback.
[in]	*FrameToCancel*	Function to select the frame to cancel; NULL to select all.
[in]	*Context*	Opaque parameters passed to FrameToCancel. Ignored if FrameToCancel is NULL.

References IP6_SERVICE_SIGNATURE, Ip6FreeNeighborEntry(), _IP6_SERVICE::Mnp, and IP6_LINK_TX_TOKEN::MnpToken.

Referenced by Ip6CancelPacket(), and Ip6CleanInterface().

```
VOID Ip6CancelPacket ( IN IP6_INTERFACE *   IpIf,
                       IN NET_BUF *         Packet,
                       IN EFI_STATUS        IoStatus
                     )
```

Cancel the Packet and all its fragments.

Parameters:

[in]	*IpIf*	The interface from which the Packet is sent.
[in]	*Packet*	The Packet to cancel.
[in]	*IoStatus*	The status returns to the sender.

References Ip6CancelFrames(), and Ip6CancelPacketFragments().

Referenced by Ip6CancelTxTokens(), Ip6Output(), and Ip6SentPacketTicking().

```
                                    IN
BOOLEAN Ip6CancelPacketFragments (  IP6_LINK_TX_TOKEN   Frame,
                                    *
                                    IN VOID *           Context
                                  )
```

The filter function to find a packet and all its fragments. The packet's fragments have their Context set to the packet.

Parameters:

[in]	*Frame*	The frames hold by the low level interface.
[in]	*Context*	Context to the function, which is the packet.

Return values:

TRUE	This is the packet to cancel or its fragments.
FALSE	This is an unrelated packet.

Referenced by Ip6CancelPacket().

```
EFI_STATUS Ip6CandidateSource ( IN IP6_SERVICE *      IpSb,
                                OUT LIST_ENTRY *      SourceList,
                                OUT UINT32 *          SourceCount
                              )
```

Output all the available source addresses to a list entry head SourceList. The number of source addresses are also returned.

Parameters:

[in]	*IpSb*	Points to an IP6 service binding instance.
[out]	*SourceList*	The list entry head of all source addresses. It is the caller's responsiblity to free the resources.
[out]	*SourceCount*	The number of source addresses.

Return values:

EFI_SUCCESS	The source addresses were copied to a list entry head SourceList.
EFI_OUT_OF_RESOURCES	Failed to allocate resources to complete the operation.

References _IP6_ADDRESS_INFO::Address, _IP6_INTERFACE::AddressList, IP6_ADDR_INFO_SIGNATURE, IP6_INFINIT_LIFETIME, IP6_LINK_LOCAL_PREFIX_LENGTH, _IP6_ADDRESS_INFO::IsAnycast, _IP6_ADDRESS_INFO::Link, _IP6_ADDRESS_INFO::PreferredLifetime, _IP6_ADDRESS_INFO::PrefixLength, _IP6_ADDRESS_INFO::Signature, and _IP6_ADDRESS_INFO::ValidLifetime.

Referenced by Ip6SelectSourceAddress().

```
UINT8 Ip6CommonPrefixLen ( IN EFI_IPv6_ADDRESS *  AddressA,
                           IN EFI_IPv6_ADDRESS *  AddressB
                         )
```

Calculate how many bits are the same between two IPv6 addresses.

Parameters:

[in]	*AddressA*	Points to an IPv6 address.
[in]	*AddressB*	Points to another IPv6 address.

Returns:

The common bits of the AddressA and AddressB.

Referenced by Ip6SelectSourceAddress().

```
EFI_STATUS Ip6Output ( IN IP6_SERVICE *          IpSb,
                       IN IP6_INTERFACE *        Interface,
                       IN IP6_PROTOCOL *         IpInstance,
                       IN NET_BUF *              Packet,
                       IN EFI_IP6_HEADER *       Head,
                       IN UINT8 *                ExtHdrs,
                       IN UINT32                 ExtHdrsLen,
                       IN IP6_FRAME_CALLBACK     Callback,
                       IN VOID *                 Context
```

)

Transmit an IP6 packet. The packet comes either from the IP6 child's consumer (IpInstance != NULL) or the IP6 driver itself (IpInstance == NULL). It will route the packet, fragment it, then transmit all the fragments through an interface.

Parameters:

[in]	*IpSb*	The IP6 service instance to transmit the packet.
[in]	*Interface*	The IP6 interface to transmit the packet. Ignored if NULL.
[in]	*IpInstance*	The IP6 child that issues the transmission. It is NULL if the packet is from the system.
[in]	*Packet*	The user data to send, excluding the IP header.
[in]	*Head*	The caller supplied header. The caller should set the following header fields: NextHeader, HopLimit, Src, Dest, FlowLabel, PayloadLength. This function will fill in the Ver, TrafficClass.
[in]	*ExtHdrs*	The extension headers to append to the IPv6 basic header.
[in]	*ExtHdrsLen*	The length of the extension headers.
[in]	*Callback*	The callback function to issue when transmission completed.
[in]	*Context*	The opaque context for the callback.

Return values:

EFI_INVALID_PARAMETER	Any input parameter or the packet is invalid.
EFI_NO_MAPPING	There is no interface to the destination.
EFI_NOT_FOUND	There is no route to the destination.
EFI_SUCCESS	The packet successfully transmitted.
EFI_OUT_OF_RESOURCES	Failed to finish the operation due to lack of resources.
Others	Failed to transmit the packet.

References IP6_ROUTE_ENTRY::Flag, _IP6_NEIGHBOR_ENTRY::Interface, IP6_GET_TICKS, IP6_PACKET_TOO_BIG, IP6_SERVICE_SIGNATURE, Ip6CancelPacket(), Ip6CreateNeighborEntry(), Ip6CreateSNMulticastAddr(), Ip6FillFragmentHeader(), Ip6FindNeighborEntry(), Ip6FindRouteEntry(), Ip6FreeRouteCacheEntry(), Ip6FreeRouteEntry(), Ip6IpSecProcessPacket(), Ip6IsAnycast(), Ip6IsExtsValid(), Ip6OnArpResolved(), Ip6PrependHead(), Ip6Route(), Ip6SelectInterface(), Ip6SelectSourceAddress(), Ip6SendFrame(), Ip6SendNeighborSolicit(), Ip6SysPacketSent(), mIp6Id, _IP6_NEIGHBOR_ENTRY::Neighbor, IP6_ROUTE_CACHE_ENTRY::NextHop, _IP6_NEIGHBOR_ENTRY::Ticks, IP6_TXTOKEN_WRAP::Token, and _IP6_NEIGHBOR_ENTRY::Transmit.

Referenced by EfiIp6Transmit(), Ip6IcmpReplyEcho(), Ip6SendIcmpError(), Ip6SendMldDone(), Ip6SendMldReport(), Ip6SendNeighborAdvertise(), Ip6SendNeighborSolicit(), and Ip6SendRouterSolicit().

```
EFI_STATUS Ip6PrependHead ( IN IP6_SERVICE *        IpSb,
                            IN NET_BUF *            Packet,
                            IN EFI_IP6_HEADER *     Head,
                            IN UINT16               FragmentOffset,
                            IN UINT8 *              ExtHdrs,
                            IN UINT32               ExtHdrsLen,
                            IN UINT8                LastHeader,
                            IN UINT32               HeadLen
                          )
```

Prefix an IP6 basic head and unfragmentable extension headers and a fragment header to the Packet. Used for IP6 fragmentation.

Parameters:

[in]	*IpSb*	The IP6 service instance to transmit the packet.
[in]	*Packet*	The packet to prefix the IP6 header to.
[in]	*Head*	The caller supplied header.
[in]	*FragmentOffset*	The fragment offset of the data following the header.
[in]	*ExtHdrs*	The length of the original extension header.
[in]	*ExtHdrsLen*	The length of the extension headers.
[in]	*LastHeader*	The pointer of next header of last extension header.
[in]	*HeadLen*	The length of the unfragmented part of the IP6 header.

Return values:

EFI_BAD_BUFFER_SIZE	There is no enought room in the head space of Packet.
EFI_SUCCESS	The operation performed successfully.

References IP6_FRAGMENT_OFFSET_MASK, and Ip6FillFragmentHeader().

Referenced by Ip6Output().

```
IP6_INTERFACE* Ip6SelectInterface ( IN IP6_SERVICE *              IpSb,
                                    IN EFI_IPv6_ADDRESS *         Destination,
                                    IN OUT EFI_IPv6_ADDRESS *     Source
                                  )
```

Select an interface to send the packet generated in the IP6 driver itself: that is, not by the requests of the IP6 child's consumer. Such packets include the ICMPv6 echo replies and other ICMPv6 error packets.

Parameters:

[in]	*IpSb*	The IP4 service that wants to send the packets.
[in]	*Destination*	The destination of the packet.
[in,out]	*Source*	The source of the packet.

Returns:

NULL if no proper interface is found, otherwise, the interface that can be used to send the system packet from.

References IP6_SERVICE_SIGNATURE, Ip6IsOneOfSetAddress(), and Ip6SelectSourceAddress().

Referenced by Ip6Output().

```
EFI_STATUS Ip6SelectSourceAddress ( IN IP6_SERVICE *              IpSb,
                                    IN EFI_IPv6_ADDRESS *         Destination,
                                    OUT EFI_IPv6_ADDRESS *        Source
                                  )
```

Output all the available source addresses to a list entry head SourceList. The number of source addresses are also returned.

Parameters:

[in]	*IpSb*	Points to a IP6 service binding instance.

[in]	*Destination*	The IPv6 destination address.
[out]	*Source*	The selected IPv6 source address according to the Destination.

Return values:

EFI_SUCCESS	The source addresses were copied to a list entry head SourceList.
EFI_NO_MAPPING	The IPv6 stack is not auto configured.

References _IP6_ADDRESS_INFO::Address, IP6_ADDR_INFO_SIGNATURE, IP6_SERVICE_SIGNATURE, Ip6CandidateSource(), Ip6CommonPrefixLen(), Ip6IsOneOfSetAddress(), Ip6RemoveAddr(), _IP6_PREFIX_LIST_ENTRY::PreferredLifetime, _IP6_PREFIX_LIST_ENTRY::Prefix, and _IP6_PREFIX_LIST_ENTRY::PrefixLength.

Referenced by EfiIp6Transmit(), Ip6ConfigProtocol(), Ip6NdFasterTimerTicking(), Ip6Output(), and Ip6SelectInterface().

```
VOID Ip6SysPacketSent ( NET_BUF *    Packet,
                        EFI_STATUS   IoStatus,
                        UINT32       LinkFlag,
                        VOID *       Context
                      )
```

The default callback function for the system generated packet. It will free the packet.

Parameters:

[in]	*Packet*	The packet that transmitted.
[in]	*IoStatus*	The result of the transmission, succeeded or failed.
[in]	*LinkFlag*	Not used when transmitted. Check IP6_FRAME_CALLBACK for reference.
[in]	*Context*	The context provided by us.

Referenced by Ip6IcmpReplyEcho(), Ip6Output(), Ip6SendIcmpError(), Ip6SendMldDone(), Ip6SendMldReport(), Ip6SendNeighborAdvertise(), Ip6SendNeighborSolicit(), and Ip6SendRouterSolicit().

Variable Documentation

UINT32 mIp6Id

Referenced by Ip6DriverBindingStart(), Ip6FillFragmentHeader(), and Ip6Output().

NetworkPkg/Ip6Dxe/Ip6Output.h File Reference

Functions

EFI_STATUS Ip6SelectSourceAddress (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *Destination, OUT EFI_IPv6_ADDRESS *Source)

VOID Ip6SysPacketSent (NET_BUF *Packet, EFI_STATUS IoStatus, UINT32 LinkFlag, VOID *Context)

EFI_STATUS Ip6Output (IN IP6_SERVICE *IpSb, IN IP6_INTERFACE *Interface, IN IP6_PROTOCOL *IpInstance, IN NET_BUF *Packet, IN EFI_IP6_HEADER *Head, IN UINT8 *ExtHdrs, IN UINT32 ExtHdrsLen, IN IP6_FRAME_CALLBACK Callback, IN VOID *Context)

VOID Ip6CancelFrames (IN IP6_INTERFACE *Interface, IN EFI_STATUS IoStatus, IN IP6_FRAME_TO_CANCEL FrameToCancel, IN VOID *Context)

VOID Ip6CancelPacket (IN IP6_INTERFACE *IpIf, IN NET_BUF *Packet, IN EFI_STATUS IoStatus)

Variables

UINT32 mIp6Id

Detailed Description

The internal functions and routines to transmit the IP6 packet.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
VOID Ip6CancelFrames ( IN IP6_INTERFACE *          Interface,
                       IN EFI_STATUS               IoStatus,
                       IN IP6_FRAME_TO_CANCEL      FrameToCancel,
                       IN VOID *                   Context
                     )
```

Remove all the frames on the interface that pass the FrameToCancel, either queued on ARP queues, or that have already been delivered to MNP and not yet recycled.

Parameters:

[in]	*Interface*	Interface to remove the frames from.
[in]	*IoStatus*	The transmit status returned to the frames' callback.
[in]	*FrameToCancel*	Function to select the frame to cancel; NULL to select all.

[in]	*Context*	Opaque parameters passed to FrameToCancel. Ignored if FrameToCancel is NULL.

Remove all the frames on the interface that pass the FrameToCancel, either queued on ARP queues or that have already been delivered to MNP and not yet recycled.

Parameters:

[in]	*Interface*	Interface to remove the frames from.
[in]	*IoStatus*	The transmit status returned to the frames' callback.
[in]	*FrameToCancel*	Function to select the frame to cancel; NULL to select all.
[in]	*Context*	Opaque parameters passed to FrameToCancel. Ignored if FrameToCancel is NULL.

References IP6_SERVICE_SIGNATURE, Ip6FreeNeighborEntry(), _IP6_SERVICE::Mnp, and IP6_LINK_TX_TOKEN::MnpToken.

Referenced by Ip6CancelPacket(), and Ip6CleanInterface().

```
VOID Ip6CancelPacket ( IN IP6_INTERFACE *   IpIf,
                       IN NET_BUF *         Packet,
                       IN EFI_STATUS        IoStatus
                     )
```

Cancel the Packet and all its fragments.

Parameters:

[in]	*IpIf*	The interface from which the Packet is sent.
[in]	*Packet*	The Packet to cancel.
[in]	*IoStatus*	The status returns to the sender.

References Ip6CancelFrames(), and Ip6CancelPacketFragments().

Referenced by Ip6CancelTxTokens(), Ip6Output(), and Ip6SentPacketTicking().

```
EFI_STATUS Ip6Output ( IN IP6_SERVICE *            IpSb,
                       IN IP6_INTERFACE *          Interface,
                       IN IP6_PROTOCOL *           IpInstance,
                       IN NET_BUF *                Packet,
                       IN EFI_IP6_HEADER *         Head,
                       IN UINT8 *                  ExtHdrs,
                       IN UINT32                   ExtHdrsLen,
                       IN IP6_FRAME_CALLBACK       Callback,
                       IN VOID *                   Context
                     )
```

Transmit an IP6 packet. The packet comes either from the IP6 child's consumer (IpInstance != NULL) or the IP6 driver itself (IpInstance == NULL). It will route the packet, fragment it, then transmit all the fragments through an interface.

Parameters:

[in]	*IpSb*	The IP6 service instance to transmit the packet.

[in]	*Interface*	The IP6 interface to transmit the packet. Ignored if NULL.
[in]	*IpInstance*	The IP6 child that issues the transmission. It is NULL if the packet is from the system.
[in]	*Packet*	The user data to send, excluding the IP header.
[in]	*Head*	The caller supplied header. The caller should set the following header fields: NextHeader, HopLimit, Src, Dest, FlowLabel, PayloadLength. This function will fill in the Ver, TrafficClass.
[in]	*ExtHdrs*	The extension headers to append to the IPv6 basic header.
[in]	*ExtHdrsLen*	The length of the extension headers.
[in]	*Callback*	The callback function to issue when transmission completed.
[in]	*Context*	The opaque context for the callback.

Return values:

EFI_INVALID_PARAMETER	Any input parameter or the packet is invalid.
EFI_NO_MAPPING	There is no interface to the destination.
EFI_NOT_FOUND	There is no route to the destination.
EFI_SUCCESS	The packet successfully transmitted.
EFI_OUT_OF_RESOURCES	Failed to finish the operation due to lack of resources.
Others	Failed to transmit the packet.

References IP6_ROUTE_ENTRY::Flag, _IP6_NEIGHBOR_ENTRY::Interface, IP6_GET_TICKS, IP6_PACKET_TOO_BIG, IP6_SERVICE_SIGNATURE, Ip6CancelPacket(), Ip6CreateNeighborEntry(), Ip6CreateSNMulticastAddr(), Ip6FillFragmentHeader(), Ip6FindNeighborEntry(), Ip6FindRouteEntry(), Ip6FreeRouteCacheEntry(), Ip6FreeRouteEntry(), Ip6IpSecProcessPacket(), Ip6IsAnycast(), Ip6IsExtsValid(), Ip6OnArpResolved(), Ip6PrependHead(), Ip6Route(), Ip6SelectInterface(), Ip6SelectSourceAddress(), Ip6SendFrame(), Ip6SendNeighborSolicit(), Ip6SysPacketSent(), mIp6Id, _IP6_NEIGHBOR_ENTRY::Neighbor, IP6_ROUTE_CACHE_ENTRY::NextHop, _IP6_NEIGHBOR_ENTRY::Ticks, IP6_TXTOKEN_WRAP::Token, and _IP6_NEIGHBOR_ENTRY::Transmit.

Referenced by EfiIp6Transmit(), Ip6IcmpReplyEcho(), Ip6SendIcmpError(), Ip6SendMldDone(), Ip6SendMldReport(), Ip6SendNeighborAdvertise(), Ip6SendNeighborSolicit(), and Ip6SendRouterSolicit().

```
EFI_STATUS Ip6SelectSourceAddress ( IN IP6_SERVICE *            IpSb,
                                    IN EFI_IPv6_ADDRESS *       Destination,
                                    OUT EFI_IPv6_ADDRESS *      Source
                                  )
```

Output all the available source addresses to the list entry head SourceList. The number of source addresses are also returned.

Parameters:

[in]	*IpSb*	Points to a IP6 service binding instance.
[in]	*Destination*	The IPv6 destination address.
[out]	*Source*	The selected IPv6 source address according to the Destination.

Return values:

EFI_SUCCESS	The source addresses were copied to the list entry head SourceList.
EFI_NO_MAPPING	The IPv6 stack is not auto configured.

Output all the available source addresses to a list entry head SourceList. The number of source addresses are also returned.

Parameters:

[in]	*IpSb*	Points to a IP6 service binding instance.
[in]	*Destination*	The IPv6 destination address.
[out]	*Source*	The selected IPv6 source address according to the Destination.

Return values:

EFI_SUCCESS	The source addresses were copied to a list entry head SourceList.
EFI_NO_MAPPING	The IPv6 stack is not auto configured.

References _IP6_ADDRESS_INFO::Address, IP6_ADDR_INFO_SIGNATURE, IP6_SERVICE_SIGNATURE, Ip6CandidateSource(), Ip6CommonPrefixLen(), Ip6IsOneOfSetAddress(), Ip6RemoveAddr(), _IP6_PREFIX_LIST_ENTRY::PreferredLifetime, _IP6_PREFIX_LIST_ENTRY::Prefix, and _IP6_PREFIX_LIST_ENTRY::PrefixLength.

Referenced by EfiIp6Transmit(), Ip6ConfigProtocol(), Ip6NdFasterTimerTicking(), Ip6Output(), and Ip6SelectInterface().

```
VOID Ip6SysPacketSent ( NET_BUF *    Packet,
                        EFI_STATUS   IoStatus,
                        UINT32       LinkFlag,
                        VOID *       Context
                      )
```

The default callback function for system generated packet. It will free the packet.

Parameters:

[in]	*Packet*	The packet that transmitted.
[in]	*IoStatus*	The result of the transmission: succeeded or failed.
[in]	*LinkFlag*	Not used when transmission. Check IP6_FRAME_CALLBACK for reference.
[in]	*Context*	The context provided by us.

The default callback function for the system generated packet. It will free the packet.

Parameters:

[in]	*Packet*	The packet that transmitted.
[in]	*IoStatus*	The result of the transmission, succeeded or failed.
[in]	*LinkFlag*	Not used when transmitted. Check IP6_FRAME_CALLBACK for reference.
[in]	*Context*	The context provided by us.

Referenced by Ip6IcmpReplyEcho(), Ip6Output(), Ip6SendIcmpError(), Ip6SendMldDone(), Ip6SendMldReport(), Ip6SendNeighborAdvertise(), Ip6SendNeighborSolicit(), and Ip6SendRouterSolicit().

Variable Documentation

UINT32 mIp6Id

Referenced by Ip6DriverBindingStart(), Ip6FillFragmentHeader(), and Ip6Output().

- Data Structures
- Files

- File List
- Globals

NetworkPkg/Ip6Dxe/Ip6Route.c File Reference

Functions

UINT32	Ip6RouteCacheHash (IN EFI_IPv6_ADDRESS *Ip1, IN EFI_IPv6_ADDRESS *Ip2)
IP6_ROUTE_ENTRY *	Ip6CreateRouteEntry (IN EFI_IPv6_ADDRESS *Destination, IN UINT8 PrefixLength, IN EFI_IPv6_ADDRESS *GatewayAddress)
VOID	Ip6FreeRouteEntry (IN OUT IP6_ROUTE_ENTRY *RtEntry)
IP6_ROUTE_ENTRY *	Ip6FindRouteEntry (IN IP6_ROUTE_TABLE *RtTable, IN EFI_IPv6_ADDRESS *Destination, IN EFI_IPv6_ADDRESS *NextHop)
IP6_ROUTE_CACHE_ENTRY *	Ip6CreateRouteCacheEntry (IN EFI_IPv6_ADDRESS *Dst, IN EFI_IPv6_ADDRESS *Src, IN EFI_IPv6_ADDRESS *GateWay, IN UINTN Tag)
VOID	Ip6FreeRouteCacheEntry (IN OUT IP6_ROUTE_CACHE_ENTRY *RtCacheEntry)
IP6_ROUTE_CACHE_ENTRY *	Ip6FindRouteCache (IN IP6_ROUTE_TABLE *RtTable, IN EFI_IPv6_ADDRESS *Dest, IN EFI_IPv6_ADDRESS *Src)
EFI_STATUS	Ip6BuildEfiRouteTable (IN IP6_ROUTE_TABLE *RouteTable, OUT UINT32 *EfiRouteCount, OUT EFI_IP6_ROUTE_TABLE **EfiRouteTable)
IP6_ROUTE_TABLE *	Ip6CreateRouteTable (VOID)
VOID	Ip6CleanRouteTable (IN OUT IP6_ROUTE_TABLE *RtTable)
VOID	Ip6PurgeRouteCache (IN IP6_ROUTE_CACHE *RtCache, IN UINTN Tag)
EFI_STATUS	Ip6AddRoute (IN OUT IP6_ROUTE_TABLE *RtTable, IN EFI_IPv6_ADDRESS *Destination, IN UINT8 PrefixLength, IN EFI_IPv6_ADDRESS *GatewayAddress)
EFI_STATUS	Ip6DelRoute (IN OUT IP6_ROUTE_TABLE *RtTable, IN EFI_IPv6_ADDRESS *Destination, IN UINT8 PrefixLength, IN EFI_IPv6_ADDRESS *GatewayAddress)
IP6_ROUTE_CACHE_ENTRY *	Ip6Route (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *Dest, IN EFI_IPv6_ADDRESS *Src)

Detailed Description

The functions and routines to handle the route caches and route table.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS Ip6AddRoute ( IN OUT IP6_ROUTE_TABLE *   RtTable,
                         IN EFI_IPv6_ADDRESS *      Destination,
                         IN UINT8                   PrefixLength,
                         IN EFI_IPv6_ADDRESS *      GatewayAddress
                       )
```

Add a route entry to the route table. It is the help function for EfiIp6Routes.

Parameters:

[in,out]	*RtTable*	Route table to add route to.
[in]	*Destination*	The destination of the network.
[in]	*PrefixLength*	The PrefixLength of the destination.
[in]	*GatewayAddress*	The next hop address.

Return values:

EFI_ACCESS_DENIED	The same route already exists.
EFI_OUT_OF_RESOURCES	Failed to allocate memory for the entry.
EFI_SUCCESS	The route was added successfully.

References IP6_ROUTE_ENTRY::Destination, IP6_ROUTE_ENTRY::Flag, IP6_DIRECT_ROUTE, Ip6CreateRouteEntry(), IP6_ROUTE_ENTRY::Link, and IP6_ROUTE_ENTRY::NextHop.

Referenced by EfiIp6Routes().

```
EFI_STATUS Ip6BuildEfiRouteTable ( IN IP6_ROUTE_TABLE *          RouteTable,
                                   OUT UINT32 *                  EfiRouteCount,
                                   OUT EFI_IP6_ROUTE_TABLE **    EfiRouteTable
                                 )
```

Build an array of EFI_IP6_ROUTE_TABLE to be returned to the caller. The number of EFI_IP6_ROUTE_TABLE is also returned.

Parameters:

[in]	*RouteTable*	The pointer of IP6_ROUTE_TABLE internal used.
[out]	*EfiRouteCount*	The number of returned route entries.
[out]	*EfiRouteTable*	The pointer to the array of EFI_IP6_ROUTE_TABLE. If NULL, only the route entry count is returned.

Return values:

EFI_SUCCESS	The EFI_IP6_ROUTE_TABLE successfully built.
EFI_OUT_OF_RESOURCES	Failed to allocate the memory for the route table.

References IP6_ROUTE_ENTRY::Destination, Ip6CopyAddressByPrefix(), IP6_ROUTE_ENTRY::NextHop, and IP6_ROUTE_ENTRY::PrefixLength.

Referenced by EfiIp6GetModeData(), and Ip6ConfigGetIfInfo().

```
VOID Ip6CleanRouteTable ( RtTable )
```

IN OUT
IP6_ROUTE_TABLE
*

Free the route table and its associated route cache. Route table is reference counted.

Parameters:

[in,out] *RtTable* The route table to free.

References IP6_ROUTE_CACHE_HASH_SIZE, Ip6FreeRouteCacheEntry(), and Ip6FreeRouteEntry().

Referenced by Ip6CleanService().

```
IP6_ROUTE_CACHE_ENTRY* Ip6CreateRouteCacheEntry ( IN EFI_IPv6_ADDRESS *  Dst,
                                                  IN EFI_IPv6_ADDRESS *  Src,
                                                  IN EFI_IPv6_ADDRESS *  GateWay,
                                                  IN UINTN               Tag
                                                )
```

Allocate and initialize a IP6 route cache entry.

Parameters:

[in]	*Dst*	The destination address.
[in]	*Src*	The source address.
[in]	*GateWay*	The next hop address.
[in]	*Tag*	The tag from the caller. This marks all the cache entries spawned from one route table entry.

Returns:

NULL if failed to allocate memory for the cache. Otherwise, point to the created route cache entry.

References IP6_ROUTE_CACHE_ENTRY::Destination, IP6_ROUTE_CACHE_ENTRY::NextHop, IP6_ROUTE_CACHE_ENTRY::RefCnt, IP6_ROUTE_CACHE_ENTRY::Source, and IP6_ROUTE_CACHE_ENTRY::Tag.

Referenced by Ip6ProcessRedirect(), and Ip6Route().

```
IP6_ROUTE_ENTRY* Ip6CreateRouteEntry ( IN EFI_IPv6_ADDRESS *  Destination,
                                       IN UINT8               PrefixLength,
                                       IN EFI_IPv6_ADDRESS *  GatewayAddress
                                     )
```

Allocate a route entry then initialize it with the Destination/PrefixLength and Gateway.

Parameters:

[in]	*Destination*	The IPv6 destination address. This is an optional parameter that may be NULL.
[in]	*PrefixLength*	The destination network's prefix length.
[in]	*GatewayAddress*	The next hop address. This is an optional parameter that may be NULL.

Returns:

NULL if failed to allocate memeory; otherwise, the newly created route entry.

References IP6_ROUTE_ENTRY::Destination, IP6_ROUTE_ENTRY::Flag, IP6_ROUTE_ENTRY::NextHop, IP6_ROUTE_ENTRY::PrefixLength, and IP6_ROUTE_ENTRY::RefCnt.

Referenced by Ip6AddRoute(), Ip6CreateDefaultRouter(), Ip6CreatePrefixListEntry(), Ip6ProcessPacketTooBig(), and Ip6ProcessRedirect().

```
IP6_ROUTE_TABLE* Ip6CreateRouteTable ( VOID )
```

Create an empty route table. This includes its internal route cache.

Returns:

NULL if failed to allocate memory for the route table. Otherwise, the point to newly created route table.

References _IP6_ROUTE_TABLE::Cache, IP6_ROUTE_CACHE::CacheBucket, IP6_ROUTE_CACHE::CacheNum, IP6_ROUTE_CACHE_HASH_SIZE, _IP6_ROUTE_TABLE::RefCnt, _IP6_ROUTE_TABLE::RouteArea, and _IP6_ROUTE_TABLE::TotalNum.

Referenced by Ip6CreateService().

```
EFI_STATUS Ip6DelRoute ( IN OUT IP6_ROUTE_TABLE *   RtTable,
                         IN EFI_IPv6_ADDRESS *      Destination,
                         IN UINT8                   PrefixLength,
                         IN EFI_IPv6_ADDRESS *      GatewayAddress
                       )
```

Remove a route entry and all the route caches spawn from it. It is the help function for EfiIp6Routes.

Parameters:

[in,out]	*RtTable*	The route table to remove the route from.
[in]	*Destination*	The destination network.
[in]	*PrefixLength*	The PrefixLength of the Destination.
[in]	*GatewayAddress*	The next hop address.

Return values:

EFI_SUCCESS	The route entry was successfully removed.
EFI_NOT_FOUND	There is no route entry in the table with that property.

References IP6_ROUTE_ENTRY::Destination, Ip6FreeRouteEntry(), Ip6PurgeRouteCache(), and IP6_ROUTE_ENTRY::NextHop.

Referenced by EfiIp6Routes(), Ip6DestroyDefaultRouter(), and Ip6DestroyPrefixListEntry().

```
IP6_ROUTE_CACHE_ENTRY* Ip6FindRouteCache ( IN IP6_ROUTE_TABLE *   RtTable,
                                           IN EFI_IPv6_ADDRESS *  Dest,
                                           IN EFI_IPv6_ADDRESS *  Src
                                         )
```

Find a route cache with the destination and source address. This is used by the ICMPv6 redirect messasge process.

Parameters:

[in]	*RtTable*	The route table to search the cache for.
[in]	*Dest*	The destination address.
[in]	*Src*	The source address.

Returns:

NULL if no route entry to the (Dest, Src). Otherwise, the pointer to the correct route cache entry.

References IP6_ROUTE_CACHE_ENTRY::Destination, IP6_ROUTE_CACHE_HASH, and IP6_ROUTE_CACHE_ENTRY::Source.

Referenced by Ip6ProcessRedirect(), and Ip6Route().

```
IP6_ROUTE_ENTRY* Ip6FindRouteEntry ( IN IP6_ROUTE_TABLE *   RtTable,
                                     IN EFI_IPv6_ADDRESS *  Destination,
                                     IN EFI_IPv6_ADDRESS *  NextHop
                                   )
```

Search the route table for a most specific match to the Dst. It searches from the longest route area (prefix length == 128) to the shortest route area (default routes). In each route area, it will first search the instance's route table, then the default route table. This is required per the following requirements: 1. IP search the route table for a most specific match. 2. The local route entries have precedence over the default route entry.

Parameters:

[in]	*RtTable*	The route table to search from.
[in]	*Destination*	The destionation address to search. If NULL, search the route table by NextHop.
[in]	*NextHop*	The next hop address. If NULL, search the route table by Destination.

Returns:

NULL if no route matches the Dst. Otherwise, the point to the

most specific route to the Dst.

References IP6_ROUTE_ENTRY::Destination, IP6_ROUTE_ENTRY::NextHop, and IP6_ROUTE_ENTRY::PrefixLength.

Referenced by Ip6Output(), Ip6ProcessPacketTooBig(), Ip6ProcessRedirect(), and Ip6Route().

```
VOID Ip6FreeRouteCacheEntry ( IN OUT IP6_ROUTE_CACHE_ENTRY *   RtCacheEntry   )
```

Free the route cache entry. It is reference counted.

Parameters:

[in,out]	*RtCacheEntry*	The route cache entry to free.

Referenced by Ip6CleanRouteTable(), Ip6NdTimerTicking(), Ip6Output(), and Ip6PurgeRouteCache().

```
VOID Ip6FreeRouteEntry ( IN OUT IP6_ROUTE_ENTRY *   RtEntry   )
```

Free the route table entry. It is reference counted.

Parameters:

[in,out] *RtEntry* The route entry to free.

Referenced by Ip6CleanRouteTable(), Ip6DelRoute(), Ip6Output(), Ip6ProcessPacketTooBig(), and Ip6Route().

```
VOID Ip6PurgeRouteCache ( IN IP6_ROUTE_CACHE *   RtCache,
                          IN UINTN               Tag
                        )
```

Remove all the cache entries bearing the Tag. When a route cache entry is created, it is tagged with the address of route entry from which it is spawned. When a route entry is deleted, the cache entries spawned from it are also deleted.

Parameters:

[in] *RtCache* Route cache to remove the entries from.

[in] *Tag* The Tag of the entries to remove.

References IP6_ROUTE_CACHE_HASH_SIZE, Ip6FreeRouteCacheEntry(), and IP6_ROUTE_CACHE_ENTRY::Tag.

Referenced by Ip6DelRoute().

```
IP6_ROUTE_CACHE_ENTRY* Ip6Route   ( IN IP6_SERVICE *        IpSb,
                                    IN
                                    EFI_IPv6_ADDRESS        Dest,
                                    *
                                    IN
                                    EFI_IPv6_ADDRESS        Src
                                    *
                                  )
```

Search the route table to route the packet. Return/create a route cache if there is a route to the destination.

Parameters:

[in] *IpSb* The IP6 service data.

[in] *Dest* The destination address to search for.

[in] *Src* The source address to search for.

Returns:

NULL if it failed to route the packet. Otherwise, a route cache entry that can be used to route packets.

References _IP6_ROUTE_TABLE::Cache, IP6_ROUTE_CACHE::CacheBucket, IP6_ROUTE_CACHE::CacheNum, IP6_ROUTE_ENTRY::Flag, IP6_DIRECT_ROUTE, IP6_ROUTE_CACHE_HASH, Ip6CreateRouteCacheEntry(), Ip6FindRouteCache(), Ip6FindRouteEntry(), Ip6FreeRouteEntry(), IP6_ROUTE_CACHE_ENTRY::Link, and IP6_ROUTE_ENTRY::NextHop.

Referenced by Ip6Output().

```
UINT32 Ip6RouteCacheHash ( IN EFI_IPv6_ADDRESS *   Ip1,
                           IN EFI_IPv6_ADDRESS *   Ip2
```

)

This is the worker function for IP6_ROUTE_CACHE_HASH(). It calculates the value as the index of the route cache bucket according to the prefix of two IPv6 addresses.

Parameters:

[in] *Ip1* The IPv6 address.

[in] *Ip2* The IPv6 address.

Returns:

The hash value of the prefix of two IPv6 addresses.

References IP6_ROUTE_CACHE_HASH_SIZE.

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Ip6Dxe/Ip6Route.h File Reference

Data Structures

struct IP6_ROUTE_ENTRY
struct IP6_ROUTE_CACHE_ENTRY
struct IP6_ROUTE_CACHE
struct _IP6_ROUTE_TABLE

Defines

"memItemRight" valign="bottom">IP6_DIRECT_ROUTE 0x00000001
#define IP6_PACKET_TOO_BIG 0x00000010
#define IP6_ROUTE_CACHE_HASH_SIZE 31
#define IP6_ROUTE_CACHE_MAX 32
#define IP6_ROUTE_CACHE_HASH(Ip1, Ip2) Ip6RouteCacheHash ((Ip1), (Ip2))

Typedefs

typedef struct _IP6_ROUTE_TABLE IP6_ROUTE_TABLE

Functions

UINT32 Ip6RouteCacheHash (IN EFI_IPv6_ADDRESS *Ip1, IN EFI_IPv6_ADDRESS *Ip2)
IP6_ROUTE_CACHE_ENTRY * Ip6CreateRouteCacheEntry (IN EFI_IPv6_ADDRESS *Dst, IN EFI_IPv6_ADDRESS *Src, IN EFI_IPv6_ADDRESS *GateWay, IN UINTN Tag)
VOID Ip6FreeRouteCacheEntry (IN OUT IP6_ROUTE_CACHE_ENTRY *RtCacheEntry)
IP6_ROUTE_CACHE_ENTRY * Ip6FindRouteCache (IN IP6_ROUTE_TABLE *RtTable, IN EFI_IPv6_ADDRESS *Dest, IN EFI_IPv6_ADDRESS *Src)
EFI_STATUS Ip6BuildEfiRouteTable (IN IP6_ROUTE_TABLE *RouteTable, OUT UINT32 *EfiRouteCount, OUT EFI_IP6_ROUTE_TABLE **EfiRouteTable)
IP6_ROUTE_TABLE * Ip6CreateRouteTable (VOID)
VOID Ip6CleanRouteTable (IN OUT IP6_ROUTE_TABLE *RtTable)
IP6_ROUTE_ENTRY * Ip6CreateRouteEntry (IN EFI_IPv6_ADDRESS *Destination, IN UINT8 PrefixLength, IN

EFI_IPv6_ADDRESS *GatewayAddress)
IP6_ROUTE_ENTRY * Ip6FindRouteEntry (IN IP6_ROUTE_TABLE *RtTable, IN EFI_IPv6_ADDRESS *Destination, IN EFI_IPv6_ADDRESS *NextHop)
VOID Ip6FreeRouteEntry (IN OUT IP6_ROUTE_ENTRY *RtEntry)
EFI_STATUS Ip6AddRoute (IN OUT IP6_ROUTE_TABLE *RtTable, IN EFI_IPv6_ADDRESS *Destination, IN UINT8 PrefixLength, IN EFI_IPv6_ADDRESS *GatewayAddress)
EFI_STATUS Ip6DelRoute (IN OUT IP6_ROUTE_TABLE *RtTable, IN EFI_IPv6_ADDRESS *Destination, IN UINT8 PrefixLength, IN EFI_IPv6_ADDRESS *GatewayAddress)
IP6_ROUTE_CACHE_ENTRY * Ip6Route (IN IP6_SERVICE *IpSb, IN EFI_IPv6_ADDRESS *Dest, IN EFI_IPv6_ADDRESS *Src)

Detailed Description

EFI IP6 route table and route cache table defintions.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Ip6AddRoute(), Ip6CreatePrefixListEntry(), Ip6ProcessPacketTooBig(), Ip6ProcessRedirect(), and Ip6Route().

#define IP6_PACKET_TOO_BIG 0x00000010

Referenced by Ip6Output(), and Ip6ProcessPacketTooBig().

#define IP6_ROUTE_CACHE_HASH (Ip1, Ip2) Ip6RouteCacheHash ((Ip1), (Ip2))

Referenced by Ip6FindRouteCache(), Ip6ProcessRedirect(), and Ip6Route().

#define IP6_ROUTE_CACHE_HASH_SIZE 31

Referenced by Ip6CleanRouteTable(), Ip6CreateRouteTable(), Ip6NdTimerTicking(), Ip6PurgeRouteCache(), and Ip6RouteCacheHash().

#define IP6_ROUTE_CACHE_MAX 32
Max NO. of cache entry per hash bucket

Referenced by Ip6NdTimerTicking().

Typedef Documentation

typedef struct _IP6_ROUTE_TABLE IP6_ROUTE_TABLE

Function Documentation

```
EFI_STATUS Ip6AddRoute ( IN OUT IP6_ROUTE_TABLE *   RtTable,
                         IN EFI_IPv6_ADDRESS *      Destination,
                         IN UINT8                   PrefixLength,
                         IN EFI_IPv6_ADDRESS *      GatewayAddress
                       )
```

Add a route entry to the route table. It is the help function for EfiIp6Routes.

Parameters:

[in,out]	*RtTable*	Route table to add route to.
[in]	*Destination*	The destination of the network.
[in]	*PrefixLength*	The PrefixLength of the destination.
[in]	*GatewayAddress*	The next hop address.

Return values:

EFI_ACCESS_DENIED	The same route already exists.
EFI_OUT_OF_RESOURCES	Failed to allocate memory for the entry.
EFI_SUCCESS	The route was added successfully.

References IP6_ROUTE_ENTRY::Destination, IP6_ROUTE_ENTRY::Flag, IP6_DIRECT_ROUTE, Ip6CreateRouteEntry(), IP6_ROUTE_ENTRY::Link, and IP6_ROUTE_ENTRY::NextHop.

Referenced by EfiIp6Routes().

```
EFI_STATUS Ip6BuildEfiRouteTable ( IN IP6_ROUTE_TABLE *          RouteTable,
                                   OUT UINT32 *                  EfiRouteCount,
                                   OUT EFI_IP6_ROUTE_TABLE **    EfiRouteTable
                                 )
```

Build a array of EFI_IP6_ROUTE_TABLE to be returned to the caller. The number of EFI_IP6_ROUTE_TABLE is also returned.

Parameters:

[in]	*RouteTable*	The pointer of IP6_ROUTE_TABLE internal used.
[out]	*EfiRouteCount*	The number of returned route entries.
[out]	*EfiRouteTable*	The pointer to the array of EFI_IP6_ROUTE_TABLE. If NULL, only the route entry count is returned.

Return values:

EFI_SUCCESS	The EFI_IP6_ROUTE_TABLE successfully built.
EFI_OUT_OF_RESOURCES	Failed to allocate the memory for the route table.

Build an array of EFI_IP6_ROUTE_TABLE to be returned to the caller. The number of EFI_IP6_ROUTE_TABLE is also returned.

Parameters:

[in]	*RouteTable*	The pointer of IP6_ROUTE_TABLE internal used.
[out]	*EfiRouteCount*	The number of returned route entries.
[out]	*EfiRouteTable*	The pointer to the array of EFI_IP6_ROUTE_TABLE. If NULL, only the route entry count is returned.

Return values:

EFI_SUCCESS	The EFI_IP6_ROUTE_TABLE successfully built.
EFI_OUT_OF_RESOURCES	Failed to allocate the memory for the route table.

References IP6_ROUTE_ENTRY::Destination, Ip6CopyAddressByPrefix(), IP6_ROUTE_ENTRY::NextHop, and IP6_ROUTE_ENTRY::PrefixLength.

Referenced by EfiIp6GetModeData(), and Ip6ConfigGetIfInfo().

```
VOID Ip6CleanRouteTable ( IN OUT IP6_ROUTE_TABLE * RtTable )
```

Free the route table and its associated route cache. Route table is reference counted.

Parameters:

[in,out]	*RtTable*	The route table to free.

References IP6_ROUTE_CACHE_HASH_SIZE, Ip6FreeRouteCacheEntry(), and Ip6FreeRouteEntry().

Referenced by Ip6CleanService().

```
IP6_ROUTE_CACHE_ENTRY* Ip6CreateRouteCacheEntry ( IN EFI_IPv6_ADDRESS * Dst,
                                                  IN EFI_IPv6_ADDRESS * Src,
                                                  IN EFI_IPv6_ADDRESS * GateWay,
                                                  IN UINTN            Tag
                                                )
```

Allocate and initialize an IP6 route cache entry.

Parameters:

[in]	*Dst*	The destination address.
[in]	*Src*	The source address.

[in]	*GateWay*	The next hop address.
[in]	*Tag*	The tag from the caller. This marks all the cache entries spawned from one route table entry.

Returns:

NULL if it failed to allocate memory for the cache. Otherwise, point to the created route cache entry.

Allocate and initialize a IP6 route cache entry.

Parameters:

[in]	*Dst*	The destination address.
[in]	*Src*	The source address.
[in]	*GateWay*	The next hop address.
[in]	*Tag*	The tag from the caller. This marks all the cache entries spawned from one route table entry.

Returns:

NULL if failed to allocate memory for the cache. Otherwise, point to the created route cache entry.

References IP6_ROUTE_CACHE_ENTRY::Destination, IP6_ROUTE_CACHE_ENTRY::NextHop, IP6_ROUTE_CACHE_ENTRY::RefCnt, IP6_ROUTE_CACHE_ENTRY::Source, and IP6_ROUTE_CACHE_ENTRY::Tag.

Referenced by Ip6ProcessRedirect(), and Ip6Route().

```
IP6_ROUTE_ENTRY* Ip6CreateRouteEntry (  IN EFI_IPv6_ADDRESS *   Destination,
                                        IN UINT8                PrefixLength,
                                        IN EFI_IPv6_ADDRESS *   GatewayAddress
                                      )
```

Allocate a route entry then initialize it with the Destination/PrefixLength and Gateway.

Parameters:

[in]	*Destination*	The IPv6 destination address. This is an optional parameter that may be NULL.
[in]	*PrefixLength*	The destination network's prefix length.
[in]	*GatewayAddress*	The next hop address. This is optional parameter that may be NULL.

Returns:

NULL if it failed to allocate memeory. Otherwise, the newly created route entry.

Allocate a route entry then initialize it with the Destination/PrefixLength and Gateway.

Parameters:

[in]	*Destination*	The IPv6 destination address. This is an optional parameter that may be NULL.
[in]	*PrefixLength*	The destination network's prefix length.
[in]	*GatewayAddress*	The next hop address. This is an optional parameter that may be NULL.

Returns:

NULL if failed to allocate memeory; otherwise, the newly created route entry.

References IP6_ROUTE_ENTRY::Destination, IP6_ROUTE_ENTRY::Flag, IP6_ROUTE_ENTRY::NextHop, IP6_ROUTE_ENTRY::PrefixLength, and IP6_ROUTE_ENTRY::RefCnt.

Referenced by Ip6AddRoute(), Ip6CreateDefaultRouter(), Ip6CreatePrefixListEntry(), Ip6ProcessPacketTooBig(), and Ip6ProcessRedirect().

IP6_ROUTE_TABLE* Ip6CreateRouteTable (VOID)

Create an empty route table, includes its internal route cache.

Returns:

NULL if failed to allocate memory for the route table. Otherwise, the point to newly created route table.

Create an empty route table. This includes its internal route cache.

Returns:

NULL if failed to allocate memory for the route table. Otherwise, the point to newly created route table.

References _IP6_ROUTE_TABLE::Cache, IP6_ROUTE_CACHE::CacheBucket, IP6_ROUTE_CACHE::CacheNum, IP6_ROUTE_CACHE_HASH_SIZE, _IP6_ROUTE_TABLE::RefCnt, _IP6_ROUTE_TABLE::RouteArea, and _IP6_ROUTE_TABLE::TotalNum.

Referenced by Ip6CreateService().

```
EFI_STATUS Ip6DelRoute ( IN OUT IP6_ROUTE_TABLE *   RtTable,
                         IN EFI_IPv6_ADDRESS *      Destination,
                         IN UINT8                   PrefixLength,
                         IN EFI_IPv6_ADDRESS *      GatewayAddress
                       )
```

Remove a route entry and all the route caches spawn from it. It is the help function for EfiIp6Routes.

Parameters:

[in,out]	*RtTable*	The route table to remove the route from.
[in]	*Destination*	The destination network.
[in]	*PrefixLength*	The PrefixLength of the Destination.
[in]	*GatewayAddress*	The next hop address.

Return values:

EFI_SUCCESS	Successfully removed the route entry.
EFI_NOT_FOUND	There is no route entry in the table with that properity.

Remove a route entry and all the route caches spawn from it. It is the help function for EfiIp6Routes.

Parameters:

[in,out]	*RtTable*	The route table to remove the route from.
[in]	*Destination*	The destination network.
[in]	*PrefixLength*	The PrefixLength of the Destination.

[in] *GatewayAddress* The next hop address.

Return values:

EFI_SUCCESS The route entry was successfully removed.
EFI_NOT_FOUND There is no route entry in the table with that property.

References IP6_ROUTE_ENTRY::Destination, Ip6FreeRouteEntry(), Ip6PurgeRouteCache(), and IP6_ROUTE_ENTRY::NextHop.

Referenced by EfiIp6Routes(), Ip6DestroyDefaultRouter(), and Ip6DestroyPrefixListEntry().

```
IP6_ROUTE_CACHE_ENTRY* Ip6FindRouteCache ( IN IP6_ROUTE_TABLE *   RtTable,
                                           IN EFI_IPv6_ADDRESS *  Dest,
                                           IN EFI_IPv6_ADDRESS *  Src
                                         )
```

Find a route cache with the destination and source address. This is used by the ICMPv6 redirect messasge process.

Parameters:

[in] *RtTable* The route table to search the cache for.
[in] *Dest* The destination address.
[in] *Src* The source address.

Returns:

NULL if no route entry to the (Dest, Src). Otherwise, point to the correct route cache entry.

Find a route cache with the destination and source address. This is used by the ICMPv6 redirect messasge process.

Parameters:

[in] *RtTable* The route table to search the cache for.
[in] *Dest* The destination address.
[in] *Src* The source address.

Returns:

NULL if no route entry to the (Dest, Src). Otherwise, the pointer to the correct route cache entry.

References IP6_ROUTE_CACHE_ENTRY::Destination, IP6_ROUTE_CACHE_HASH, and IP6_ROUTE_CACHE_ENTRY::Source.

Referenced by Ip6ProcessRedirect(), and Ip6Route().

```
IP6_ROUTE_ENTRY* Ip6FindRouteEntry ( IN IP6_ROUTE_TABLE *   RtTable,
                                     IN EFI_IPv6_ADDRESS *  Destination,
                                     IN EFI_IPv6_ADDRESS *  NextHop
                                   )
```

Search the route table for a most specific match to the Dst. It searches from the longest route area (prefix length == 128) to the shortest route area (default routes). In each route area, it will first search the instance's route table, then the default route table. This is required per the following requirements: 1. IP search the route table for a most specific match. 2. The local route entries have precedence over the default route entry.

Parameters:

[in] *RtTable* The route table to search from.
[in] *Destination* The destionation address to search. If NULL, search the route table by NextHop.
[in] *NextHop* The next hop address. If NULL, search the route table by Destination.

Returns:

NULL if no route matches the Dst. Otherwise the point to the most specific route to the Dst.

Search the route table for a most specific match to the Dst. It searches from the longest route area (prefix length == 128) to the shortest route area (default routes). In each route area, it will first search the instance's route table, then the default route table. This is required per the following requirements: 1. IP search the route table for a most specific match. 2. The local route entries have precedence over the default route entry.

Parameters:

[in] *RtTable* The route table to search from.
[in] *Destination* The destionation address to search. If NULL, search the route table by NextHop.
[in] *NextHop* The next hop address. If NULL, search the route table by Destination.

Returns:

NULL if no route matches the Dst. Otherwise, the point to the

most specific route to the Dst.

References IP6_ROUTE_ENTRY::Destination, IP6_ROUTE_ENTRY::NextHop, and IP6_ROUTE_ENTRY::PrefixLength.

Referenced by Ip6Output(), Ip6ProcessPacketTooBig(), Ip6ProcessRedirect(), and Ip6Route().

VOID Ip6FreeRouteCacheEntry (IN OUT IP6_ROUTE_CACHE_ENTRY * *RtCacheEntry*)

Free the route cache entry. It is reference counted.

Parameters:

[in,out] *RtCacheEntry* The route cache entry to free.

Referenced by Ip6CleanRouteTable(), Ip6NdTimerTicking(), Ip6Output(), and Ip6PurgeRouteCache().

VOID Ip6FreeRouteEntry (IN OUT IP6_ROUTE_ENTRY * *RtEntry*)

Free the route table entry. It is reference counted.

Parameters:

[in,out] *RtEntry* The route entry to free.

Referenced by Ip6CleanRouteTable(), Ip6DelRoute(), Ip6Output(), Ip6ProcessPacketTooBig(), and Ip6Route().

IP6_ROUTE_CACHE_ENTRY* Ip6Route (IN IP6_SERVICE * *IpSb*,
IN EFI_IPv6_ADDRESS * *Dest*,
Src

IN
EFI_IPv6_ADDRESS
*
)

Search the route table to route the packet. Return/create a route cache if there is a route to the destination.

Parameters:

[in] *IpSb* The IP6 service data.
[in] *Dest* The destination address to search for.
[in] *Src* The source address to search for.

Returns:

NULL if failed to route packet. Otherwise, a route cache entry that can be used to route packet.

Search the route table to route the packet. Return/create a route cache if there is a route to the destination.

Parameters:

[in] *IpSb* The IP6 service data.
[in] *Dest* The destination address to search for.
[in] *Src* The source address to search for.

Returns:

NULL if it failed to route the packet. Otherwise, a route cache entry that can be used to route packets.

References _IP6_ROUTE_TABLE::Cache, IP6_ROUTE_CACHE::CacheBucket, IP6_ROUTE_CACHE::CacheNum, IP6_ROUTE_ENTRY::Flag, IP6_DIRECT_ROUTE, IP6_ROUTE_CACHE_HASH, Ip6CreateRouteCacheEntry(), Ip6FindRouteCache(), Ip6FindRouteEntry(), Ip6FreeRouteEntry(), IP6_ROUTE_CACHE_ENTRY::Link, and IP6_ROUTE_ENTRY::NextHop.

Referenced by Ip6Output().

UINT32 Ip6RouteCacheHash (IN EFI_IPv6_ADDRESS * *Ip1*,
IN EFI_IPv6_ADDRESS * *Ip2*
)

This is the worker function for IP6_ROUTE_CACHE_HASH(). It calculates the value as the index of the route cache bucket according to the prefix of two IPv6 addresses.

Parameters:

[in] *Ip1* The IPv6 address.
[in] *Ip2* The IPv6 address.

Returns:

The hash value of the prefix of two IPv6 addresses.

References IP6_ROUTE_CACHE_HASH_SIZE.

- File List
- Globals

NetworkPkg/IpSecDxe/ComponentName.c File Reference

Functions

EFI_STATUS EFIAPI IpSecComponentNameGetDriverName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN CHAR8 *Language, OUT CHAR16 **DriverName)

EFI_STATUS EFIAPI IpSecComponentNameGetControllerName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE ChildHandle, IN CHAR8 *Language, OUT CHAR16 **ControllerName)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gIpSecComponentName

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gIpSecComponentName2

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mIpSecDriverNameTable []

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mIpSecControllerNameTable []

Detailed Description

UEFI Component Name(2) protocol implementation for IPsec driver.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI IpSecComponentNameGetControllerName (	IN EFI_COMPONENT_NAME_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_HANDLE	*ChildHandle*,
	IN CHAR8 *	*Language*,

OUT CHAR16 ** *ControllerName*
)

Retrieves a Unicode string that is the user-readable name of the controller that is being managed by a driver.

This function retrieves the user-readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

`[in]`	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
`[in]`	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
`[in]`	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
`[in]`	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
`[out]`	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

Retrieves a Unicode string that is the user-readable name of the controller that is being managed by a driver.

This function retrieves the user-readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
[in]	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL, and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gIpSecComponentName, and mIpSecControllerNameTable.

```
EFI_STATUS EFIAPI
IpSecComponentNameGetDriverName
  (
    IN EFI_COMPONENT_NAME_PROTOCOL *   This,
    IN CHAR8 *                         Language,
    OUT CHAR16 **                      DriverName
  )
```

Retrieves a Unicode string that is the user-readable name of the driver.

This function retrieves the user-readable name of a driver in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the driver name is returned in DriverName, and EFI_SUCCESS is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.

[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*DriverName*	A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by This in the language specified by Language.

Return values:

EFI_SUCCESS	The Unicode string for the Driver specified by This and the language specified by Language was returned in DriverName.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	DriverName is NULL.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

Retrieves a Unicode string that is the user-readable name of the driver.

This function retrieves the user-readable name of a driver in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the driver name is returned in DriverName, and EFI_SUCCESS is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*DriverName*	A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by This in the language specified by Language.

Return values:

EFI_SUCCESS	The Unicode string for the Driver specified by This, and the language specified by Language was returned in DriverName.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	DriverName is NULL.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gIpSecComponentName, and mIpSecDriverNameTable.

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL
gIpSecComponentName

Initial value:

```
{
 IpSecComponentNameGetDriverName,
 IpSecComponentNameGetControllerName,
```

```
  "eng"
}
```

Referenced by IpSecComponentNameGetControllerName(), IpSecComponentNameGetDriverName(), and IpSecDriverEntryPoint().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gIpSecComponentName2

Initial value:

```
 {
  (EFI_COMPONENT_NAME2_GET_DRIVER_NAME) IpSecComponentNameGetDriverName,
  (EFI_COMPONENT_NAME2_GET_CONTROLLER_NAME) IpSecComponentNameGetControllerName,
  "en"
}
```

Referenced by IpSecDriverEntryPoint().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mIpSecControllerNameTable[]

Initial value:

```
 {
  {
    "eng;en",
    L"IPsec Controller"
  },
  {
    NULL,
    NULL
  }
}
```

Referenced by IpSecComponentNameGetControllerName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mIpSecDriverNameTable[]

Initial value:

```
 {
  {
    "eng;en",
    L"IpSec Driver"
  },
  {
    NULL,
    NULL
  }
}
```

Referenced by IpSecComponentNameGetDriverName().

- Globals

NetworkPkg/IpSecDxe/IetfConstants.c File Reference

Defines

"memItemRight" valign="bottom">DH_GENERATOR_2 2

Variables

GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp768Modulus []
GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp1024Modulus []
GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp1536Modulus []
GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp2048Modulus []
GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp3072Modulus []
GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp4096Modulus []
GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp6144Modulus []
GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp8192Modulus []
GLOBAL_REMOVE_IF_UNREFERENCED CONST MODP_GROUP OakleyModpGroup []

Detailed Description

Cryptographic Parameter Constant Definitions from IETF;

Copyright (c) 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp1024Modulus[]

Initial value:

```
{
 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC9,0x0F,0xDA,0xA2,
 0x21,0x68,0xC2,0x34,0xC4,0xC6,0x62,0x8B,0x80,0xDC,0x1C,0xD1,
 0x29,0x02,0x4E,0x08,0x8A,0x67,0xCC,0x74,0x02,0x0B,0xBE,0xA6,
 0x3B,0x13,0x9B,0x22,0x51,0x4A,0x08,0x79,0x8E,0x34,0x04,0xDD,
```

```
  0xEF,0x95,0x19,0xB3,0xCD,0x3A,0x43,0x1B,0x30,0x2B,0x0A,0x6D,
  0xF2,0x5F,0x14,0x37,0x4F,0xE1,0x35,0x6D,0x6D,0x51,0xC2,0x45,
  0xE4,0x85,0xB5,0x76,0x62,0x5E,0x7E,0xC6,0xF4,0x4C,0x42,0xE9,
  0xA6,0x37,0xED,0x6B,0x0B,0xFF,0x5C,0xB6,0xF4,0x06,0xB7,0xED,
  0xEE,0x38,0x6B,0xFB,0x5A,0x89,0x9F,0xA5,0xAE,0x9F,0x24,0x11,
  0x7C,0x4B,0x1F,0xE6,0x49,0x28,0x66,0x51,0xEC,0xE6,0x53,0x81,
  0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
  }
```

GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp1536Modulus[]

Initial value:

```
{
  0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC9,0x0F,0xDA,0xA2,
  0x21,0x68,0xC2,0x34,0xC4,0xC6,0x62,0x8B,0x80,0xDC,0x1C,0xD1,
  0x29,0x02,0x4E,0x08,0x8A,0x67,0xCC,0x74,0x02,0x0B,0xBE,0xA6,
  0x3B,0x13,0x9B,0x22,0x51,0x4A,0x08,0x79,0x8E,0x34,0x04,0xDD,
  0xEF,0x95,0x19,0xB3,0xCD,0x3A,0x43,0x1B,0x30,0x2B,0x0A,0x6D,
  0xF2,0x5F,0x14,0x37,0x4F,0xE1,0x35,0x6D,0x6D,0x51,0xC2,0x45,
  0xE4,0x85,0xB5,0x76,0x62,0x5E,0x7E,0xC6,0xF4,0x4C,0x42,0xE9,
  0xA6,0x37,0xED,0x6B,0x0B,0xFF,0x5C,0xB6,0xF4,0x06,0xB7,0xED,
  0xEE,0x38,0x6B,0xFB,0x5A,0x89,0x9F,0xA5,0xAE,0x9F,0x24,0x11,
  0x7C,0x4B,0x1F,0xE6,0x49,0x28,0x66,0x51,0xEC,0xE4,0x5B,0x3D,
  0xC2,0x00,0x7C,0xB8,0xA1,0x63,0xBF,0x05,0x98,0xDA,0x48,0x36,
  0x1C,0x55,0xD3,0x9A,0x69,0x16,0x3F,0xA8,0xFD,0x24,0xCF,0x5F,
  0x83,0x65,0x5D,0x23,0xDC,0xA3,0xAD,0x96,0x1C,0x62,0xF3,0x56,
  0x20,0x85,0x52,0xBB,0x9E,0xD5,0x29,0x07,0x70,0x96,0x96,0x6D,
  0x67,0x0C,0x35,0x4E,0x4A,0xBC,0x98,0x04,0xF1,0x74,0x6C,0x08,
  0xCA,0x23,0x73,0x27,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
  }
```

GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp2048Modulus[]

Initial value:

```
{
  0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC9,0x0F,0xDA,0xA2,
  0x21,0x68,0xC2,0x34,0xC4,0xC6,0x62,0x8B,0x80,0xDC,0x1C,0xD1,
  0x29,0x02,0x4E,0x08,0x8A,0x67,0xCC,0x74,0x02,0x0B,0xBE,0xA6,
  0x3B,0x13,0x9B,0x22,0x51,0x4A,0x08,0x79,0x8E,0x34,0x04,0xDD,
  0xEF,0x95,0x19,0xB3,0xCD,0x3A,0x43,0x1B,0x30,0x2B,0x0A,0x6D,
  0xF2,0x5F,0x14,0x37,0x4F,0xE1,0x35,0x6D,0x6D,0x51,0xC2,0x45,
  0xE4,0x85,0xB5,0x76,0x62,0x5E,0x7E,0xC6,0xF4,0x4C,0x42,0xE9,
  0xA6,0x37,0xED,0x6B,0x0B,0xFF,0x5C,0xB6,0xF4,0x06,0xB7,0xED,
  0xEE,0x38,0x6B,0xFB,0x5A,0x89,0x9F,0xA5,0xAE,0x9F,0x24,0x11,
  0x7C,0x4B,0x1F,0xE6,0x49,0x28,0x66,0x51,0xEC,0xE4,0x5B,0x3D,
  0xC2,0x00,0x7C,0xB8,0xA1,0x63,0xBF,0x05,0x98,0xDA,0x48,0x36,
  0x1C,0x55,0xD3,0x9A,0x69,0x16,0x3F,0xA8,0xFD,0x24,0xCF,0x5F,
  0x83,0x65,0x5D,0x23,0xDC,0xA3,0xAD,0x96,0x1C,0x62,0xF3,0x56,
  0x20,0x85,0x52,0xBB,0x9E,0xD5,0x29,0x07,0x70,0x96,0x96,0x6D,
  0x67,0x0C,0x35,0x4E,0x4A,0xBC,0x98,0x04,0xF1,0x74,0x6C,0x08,
  0xCA,0x18,0x21,0x7C,0x32,0x90,0x5E,0x46,0x2E,0x36,0xCE,0x3B,
  0xE3,0x9E,0x77,0x2C,0x18,0x0E,0x86,0x03,0x9B,0x27,0x83,0xA2,
  0xEC,0x07,0xA2,0x8F,0xB5,0xC5,0x5D,0xF0,0x6F,0x4C,0x52,0xC9,
  0xDE,0x2B,0xCB,0xF6,0x95,0x58,0x17,0x18,0x39,0x95,0x49,0x7C,
  0xEA,0x95,0x6A,0xE5,0x15,0xD2,0x26,0x18,0x98,0xFA,0x05,0x10,
  0x15,0x72,0x8E,0x5A,0x8A,0xAC,0xAA,0x68,0xFF,0xFF,0xFF,0xFF,
  0xFF,0xFF,0xFF,0xFF,
  }
```

GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp3072Modulus[]

GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp4096Modulus[]

GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp6144Modulus[]

GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp768Modulus[]

Initial value:

```
{
 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xC9, 0x0F, 0xDA, 0xA2,
 0x21, 0x68, 0xC2, 0x34, 0xC4, 0xC6, 0x62, 0x8B, 0x80, 0xDC, 0x1C, 0xD1,
 0x29, 0x02, 0x4E, 0x08, 0x8A, 0x67, 0xCC, 0x74, 0x02, 0x0B, 0xBE, 0xA6,
 0x3B, 0x13, 0x9B, 0x22, 0x51, 0x4A, 0x08, 0x79, 0x8E, 0x34, 0x04, 0xDD,
 0xEF, 0x95, 0x19, 0xB3, 0xCD, 0x3A, 0x43, 0x1B, 0x30, 0x2B, 0x0A, 0x6D,
 0xF2, 0x5F, 0x14, 0x37, 0x4F, 0xE1, 0x35, 0x6D, 0x6D, 0x51, 0xC2, 0x45,
 0xE4, 0x85, 0xB5, 0x76, 0x62, 0x5E, 0x7E, 0xC6, 0xF4, 0x4C, 0x42, 0xE9,
 0xA6, 0x3A, 0x36, 0x20, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
 }
```

GLOBAL_REMOVE_IF_UNREFERENCED UINT8 Modp8192Modulus[]

GLOBAL_REMOVE_IF_UNREFERENCED CONST MODP_GROUP OakleyModpGroup[]

Initial value:

```
{
 {0,                      0,    NULL,              0},
 {OakleyGroupModp768,   768,  Modp768Modulus,   DH_GENERATOR_2},
 {OakleyGroupModp1024, 1024, Modp1024Modulus, DH_GENERATOR_2},
 {0,                      0,    NULL,              0},
 {0,                      0,    NULL,              0},
 {OakleyGroupModp1536, 1536, Modp1536Modulus, DH_GENERATOR_2},
 {0,                      0,    NULL,              0},
 {0,                      0,    NULL,              0},
 {0,                      0,    NULL,              0},
 {0,                      0,    NULL,              0},
 {0,                      0,    NULL,              0},
 {0,                      0,    NULL,              0},
 {0,                      0,    NULL,              0},
 {0,                      0,    NULL,              0},
 {OakleyGroupModp2048, 2048, Modp2048Modulus, DH_GENERATOR_2},
 {OakleyGroupModp3072, 3072, Modp3072Modulus, DH_GENERATOR_2},
 {OakleyGroupModp4096, 4096, Modp4096Modulus, DH_GENERATOR_2},
 {OakleyGroupModp6144, 6144, Modp6144Modulus, DH_GENERATOR_2},
 {OakleyGroupModp8192, 8192, Modp8192Modulus, DH_GENERATOR_2},
}
```

Referenced by Ikev2GenerateSaDhPublicKey().

NetworkPkg/IpSecDxe/Ike.h File Reference

Data Structures

struct IKE_HEADER
union IKE_SA_ATTR_UNION
struct IKE_SA_ATTRIBUTE
struct IKE_PACKET
struct IKE_PAYLOAD
struct IKE_UDP_SERVICE
struct SA_KEYMATS
struct IKE_ALG_GUID_INFO
struct MODP_GROUP

Defines

"memItemRight" valign="bottom">IKE_VERSION_MAJOR_MASK 0xf0

#define IKE_VERSION_MINOR_MASK 0x0f
#define IKE_MAJOR_VERSION(v) (((v) & IKE_VERSION_MAJOR_MASK) >> 4)
#define IKE_MINOR_VERSION(v) ((v) & IKE_VERSION_MINOR_MASK)
#define IPSEC_PROTO_ISAKMP 1
#define IPSEC_PROTO_IPSEC_AH 2
#define IPSEC_PROTO_IPSEC_ESP 3
#define IPSEC_PROTO_IPCOMP 4
#define IKE_ENCRYPT_TYPE 0
#define IKE_AUTH_TYPE 1
#define IKE_PRF_TYPE 2
#define IKE_DH_TYPE 3
#define IPSEC_ESP_DES_IV64 1
#define IPSEC_ESP_DES 2
#define IPSEC_ESP_3DES 3
#define IPSEC_ESP_RC5 4
#define IPSEC_ESP_IDEA 5
#define IPSEC_ESP_CAST 6
#define IPSEC_ESP_BLOWFISH 7
#define IPSEC_ESP_3IDEA 8
#define IPSEC_ESP_DES_IV32 9
#define IPSEC_ESP_RC4 10
#define IPSEC_ESP_NULL 11
#define IPSEC_ESP_AES 12
#define IKE_XCG_TYPE_NONE 0
#define IKE_XCG_TYPE_BASE 1
#define IKE_XCG_TYPE_IDENTITY_PROTEC
#define IKE_XCG_TYPE_AUTH_ONLY 3
#define IKE_XCG_TYPE_AGGR 4
#define IKE_XCG_TYPE_INFO 5
#define IKE_XCG_TYPE_QM 32
#define IKE_XCG_TYPE_NGM 33
#define IKE_XCG_TYPE_SA_INIT 34
#define IKE_XCG_TYPE_AUTH 35

```
#define IKE_XCG_TYPE_CREATE_CHILD_SA   36
#define IKE_XCG_TYPE_INFO2   37
#define IKE_LIFE_TYPE_SECONDS   1
#define IKE_LIFE_TYPE_KILOBYTES   2
#define IKE_SA_DEFAULT_LIFETIME   1200
#define CHILD_SA_DEFAULT_LIFETIME   3600
#define IKE_PROPOSAL_NEXT_PAYLOAD_MORE
#define IKE_PROPOSAL_NEXT_PAYLOAD_NONE
#define IKE_TRANSFORM_NEXT_PAYLOAD_MORE
#define IKE_TRANSFORM_NEXT_PAYLOAD_NONE
#define MAX_SA_ATTRS_SIZE   48
#define SA_ATTR_FORMAT_BIT   0x8000
#define INFO_MID_SIGNATURE   SIGNATURE_32 ('
        'N', 'F', 'M')
```

Typedefs

```
typedef VOID(* IKE_ON_PAYLOAD_FROM_NET )(IN UINT8
        *SessionCommon, IN UINT8 *PayloadBuf, IN
        UINTN PayloadSize, IN UINT8 PayloadType)
```

Enumerations

```
enum  IKE_SESSION_TYPE { IkeSessionTypeIkeSa,
      IkeSessionTypeChildSa, IkeSessionTypeInfo,
      IkeSessionTypeMax }
enum  OAKLEY_GROUP_ID {
        OakleyGroupModp768 = 1,
      OakleyGroupModp1024 = 2, OakleyGroupGp15
      3, OakleyGroupGp185 = 4,
        OakleyGroupModp1536 = 5,
      OakleyGroupModp2048 = 14,
      OakleyGroup Modp3072 = 15,
      OakleyGroupModp4096 = 16,
        OakleyGroupModp6144 = 17,
      OakleyGroupModp8192 = 18, OakleyGroupMax
      }
```

Detailed Description

The common definition of IPsec Key Exchange (IKE).

Copyright (c) 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Ikev2ChildSaSessionReg().

#define IKE_AUTH_TYPE 1

Referenced by Ikev2IsSupportAlg(), and Ikev2ParseProposalData().

#define IKE_DH_TYPE 3

Referenced by Ikev2IsSupportAlg(), and Ikev2ParseProposalData().

#define IKE_ENCRYPT_TYPE 0

Referenced by Ikev2IsSupportAlg(), and Ikev2ParseProposalData().

#define IKE_LIFE_TYPE_KILOBYTES 2

#define IKE_LIFE_TYPE_SECONDS 1

#define IKE_MAJOR_VERSION (v) (((v) & IKE_VERSION_MAJOR_MASK) >> 4)

Referenced by IkeDispatch().

#define IKE_MINOR_VERSION (v) ((v) & IKE_VERSION_MINOR_MASK)

#define IKE_PRF_TYPE 2

Referenced by Ikev2IsSupportAlg(), and Ikev2ParseProposalData().

#define IKE_PROPOSAL_NEXT_PAYLOAD_MORE 2

Referenced by Ikev2DecodeSa(), and Ikev2EncodeSa().

#define IKE_PROPOSAL_NEXT_PAYLOAD_NONE 0

Referenced by Ikev2EncodeSa().

#define IKE_SA_DEFAULT_LIFETIME 1200

Referenced by Ikev2SaSessionReg().

#define IKE_TRANSFORM_NEXT_PAYLOAD_MORE 3

Referenced by Ikev2EncodeSa().

#define IKE_TRANSFORM_NEXT_PAYLOAD_NONE 0

Referenced by Ikev2EncodeSa().

#define IKE_VERSION_MAJOR_MASK 0xf0

#define IKE_VERSION_MINOR_MASK 0x0f

#define IKE_XCG_TYPE_AGGR 4

#define IKE_XCG_TYPE_AUTH 35

Referenced by IkeDispatch(), and IpSecDumpPacket().

#define IKE_XCG_TYPE_AUTH_ONLY 3

#define IKE_XCG_TYPE_BASE 1

#define IKE_XCG_TYPE_CREATE_CHILD_SA 36

Referenced by IkeDispatch(), Ikev2CreateChildGenerator(), and IpSecDumpPacket().

#define IKE_XCG_TYPE_IDENTITY_PROTECT 2

Referenced by IkeDispatch(), and IpSecDumpPacket().

#define IKE_XCG_TYPE_INFO 5

Referenced by IkeDispatch(), and IpSecDumpPacket().

#define IKE_XCG_TYPE_INFO2 37

Referenced by IkeDispatch(), and IpSecDumpPacket().

#define IKE_XCG_TYPE_NGM 33

#define IKE_XCG_TYPE_NONE 0

#define IKE_XCG_TYPE_QM 32

Referenced by IkeDispatch(), and IpSecDumpPacket().

#define IKE_XCG_TYPE_SA_INIT 34

Referenced by IkeDispatch(), and IpSecDumpPacket().

#define INFO_MID_SIGNATURE SIGNATURE_32 ('I', 'N', 'F', 'M')

#define IPSEC_ESP_3DES 3

#define IPSEC_ESP_3IDEA 8

#define IPSEC_ESP_AES 12

#define IPSEC_ESP_BLOWFISH 7

#define IPSEC_ESP_CAST 6

#define IPSEC_ESP_DES 2

#define IPSEC_ESP_DES_IV32 9

#define IPSEC_ESP_DES_IV64 1

#define IPSEC_ESP_IDEA 5

#define IPSEC_ESP_NULL 11

#define IPSEC_ESP_RC4 10

#define IPSEC_ESP_RC5 4

#define IPSEC_PROTO_IPCOMP 4

#define IPSEC_PROTO_IPSEC_AH 2

#define IPSEC_PROTO_IPSEC_ESP 3

Referenced by Ikev2ChildSaParseSaPayload(), Ikev2GenerateDeletePayload(), and Ikev2InitializeSaData().

#define IPSEC_PROTO_ISAKMP 1

Referenced by Ikev2GenerateDeletePayload(), Ikev2InitializeSaData(), Ikev2InitPskGenerator(), Ikev2ParserNotifyCookiePayload(), and Ikev2SaParseSaPayload().

#define MAX_SA_ATTRS_SIZE 48

Referenced by Ikev2EncodeSa().

#define SA_ATTR_FORMAT_BIT 0x8000

Referenced by Ikev2DecodeSa(), and Ikev2EncodeSa().

Typedef Documentation

typedef VOID(* IKE_ON_PAYLOAD_FROM_NET)(IN UINT8 *SessionCommon, IN UINT8 *PayloadBuf, IN UINTN PayloadSize, IN UINT8 PayloadType)

This is prototype definition of general interface to phase the payloads after/before the decode/encode.

Parameters:

[in]	*SessionCommon*	Point to the SessionCommon
[in]	*PayloadBuf*	Point to the buffer of Payload.
[in]	*PayloadSize*	The size of the PayloadBuf in bytes.
[in]	*PayloadType*	The type of Payload.

Enumeration Type Documentation

enum IKE_SESSION_TYPE

Enumerator:

IkeSessionTypeIkeSa
IkeSessionTypeChildSa
IkeSessionTypeInfo
IkeSessionTypeMax

enum OAKLEY_GROUP_ID

Enumerator:

OakleyGroupModp768
OakleyGroupModp1024
OakleyGroupGp155
OakleyGroupGp185
OakleyGroupModp1536
OakleyGroupModp2048
OakleyGroupModp3072
OakleyGroupModp4096
OakleyGroupModp6144
OakleyGroupModp8192
OakleyGroupMax

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/IkeCommon.c File Reference

Functions

UINT64 IkeGenerateCookie (VOID)
UINT8 * IkeGenerateNonce (IN UINTN NonceSize)
VOID IkeHdrNetToHost (IN OUT IKE_HEADER *Header)
VOID IkeHdrHostToNet (IN OUT IKE_HEADER *Header)
IKE_PAYLOAD * IkePayloadAlloc (VOID)
VOID IkePayloadFree (IN IKE_PAYLOAD *IkePayload)
UINT32 IkeGenerateSpi (VOID)
EFI_STATUS IkeGenerateIv (IN UINT8 *IvBuffer, IN UINTN IvSize)
IPSEC_SPD_ENTRY * IkeSearchSpdEntry (IN EFI_IPSEC_SPD_SELECTOR *SpdSel)
UINT8 IkeGetVersionFromSession (IN UINT8 *Session)

Variables

UINT32 mNextSpi = IKE_SPI_BASE

Detailed Description

Common operation of the IKE

Copyright (c) 2010 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

UINT64 IkeGenerateCookie (VOID)

Call Crypto Lib to generate a random value with eight-octet length.

Returns:

the 64 byte vaule.

References IpSecCryptoIoGenerateRandomBytes().

Referenced by Ikev2InitPskParser(), and Ikev2SaSessionAlloc().

EFI_STATUS IkeGenerateIv (IN UINT8 * *IvBuffer*,

IN UINTN *IvSize*
)

Generate a random data for IV

Parameters:

[in] *IvBuffer* The pointer of the IV buffer.
[in] *IvSize* The IV size.

Return values:

EFI_SUCCESS Create a random data for IV.
otherwise Failed.

References IpSecCryptoIoGenerateRandomBytes().

Referenced by Ikev2EncryptPacket().

UINT8* IkeGenerateNonce (IN UINTN *NonceSize*)

Generate the random data for Nonce payload.

Parameters:

[in] *NonceSize* Size of the data in bytes.

Returns:

Buffer which contains the random data of the spcified size.

References IpSecCryptoIoGenerateRandomBytes().

Referenced by Ikev2InitPskGenerator(), and Ikev2InitPskParser().

UINT32 IkeGenerateSpi (VOID)

Generate an new SPI.

Returns:

a SPI in 4 bytes.

References mNextSpi.

Referenced by Ikev2ChildSaSessionAlloc().

UINT8 IkeGetVersionFromSession (IN UINT8 * *Session*)

Get the IKE Version from the IKE_SA_SESSION.

Parameters:

[in] *Session* Pointer of the IKE_SA_SESSION.

References IKEV2_SA_SESSION_SIGNATURE.

Referenced by IkeDeleteAllSas(), and IkeNegotiate().

VOID IkeHdrHostToNet (IN OUT IKE_HEADER * *Header*)
Convert the IKE Header from Host order to Network order.

Parameters:
[in,out] *Header* The pointer of the IKE_HEADER.

Referenced by Ikev2DecryptPacket(), Ikev2EncodePacket(), and Ikev2EncryptPacket().

VOID IkeHdrNetToHost (IN OUT IKE_HEADER * *Header*)
Convert the IKE Header from Network order to Host order.

Parameters:
[in,out] *Header* The pointer of the IKE_HEADER.

Referenced by IkePacketFromNetbuf(), and Ikev2DecodePacket().

IKE_PAYLOAD* IkePayloadAlloc (VOID)
Allocate a buffer of IKE_PAYLOAD and set its Signature.

Returns:
A buffer of IKE_PAYLOAD.

References IKE_PAYLOAD_SIGNATURE, and IKE_PAYLOAD::Signature.

Referenced by Ikev2CertGenerateAuthPayload(), Ikev2DecodePacket(), Ikev2EncryptPacket(), Ikev2GenerateCertIdPayload(), Ikev2GenerateCertificatePayload(), Ikev2GenerateCpPayload(), Ikev2GenerateDeletePayload(), Ikev2GenerateIdPayload(), Ikev2GenerateKePayload(), Ikev2GenerateNoncePayload(), Ikev2GenerateNotifyPayload(), Ikev2GenerateSaPayload(), Ikev2GenerateTsPayload(), and Ikev2PskGenerateAuthPayload().

VOID IkePayloadFree (IN IKE_PAYLOAD * *IkePayload*)
Free a specified IKE_PAYLOAD buffer.

Parameters:
[in] *IkePayload* Pointer of IKE_PAYLOAD to be freed.

Referenced by ClearAllPayloads(), IkePacketFree(), Ikev2AuthCertParser(), Ikev2CertGenerateAuthPayload(), Ikev2GenerateTsPayload(), Ikev2InitPskGenerator(), and Ikev2PskGenerateAuthPayload().

IPSEC_SPD_ENTRY* IkeSearchSpdEntry (IN EFI_IPSEC_SPD_SELECTOR * *SpdSel*)
Find SPD entry by a specified SPD selector.

Parameters:
[in] *SpdSel* Point to SPD Selector to be searched for.

Return values:
Point to SPD Entry if the SPD entry found.
NULL if not found.

References CompareSpdSelector(), IPSEC_SPD_ENTRY_FROM_LIST, mConfigData, and _IPSEC_SPD_ENTRY::Selector.

Referenced by Ikev2MatchSpdEntry().

Variable Documentation

UINT32 mNextSpi = IKE_SPI_BASE

Referenced by IkeGenerateSpi().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/IkeCommon.h File Reference

Defines

"memItemRight" valign="bottom">IKE_DEFAULT_PORT 500

#define IKE_DEFAULT_TIMEOUT_INTERVAL 10000

#define IKE_NONCE_SIZE 16

#define IKE_MAX_RETRY 4

#define IKE_SPI_BASE 0x10000

#define IKE_PAYLOAD_SIGNATURE SIGNATURE_32

#define IKE_PAYLOAD_BY_PACKET(a) CR(a,IKE_PA

#define IKE_PACKET_APPEND_PAYLOAD(IkePacket, I

#define IKE_PACKET_REMOVE_PAYLOAD(IkePacket,

#define IKE_PACKET_END_PAYLOAD(IkePacket, Node

Functions

UINT64 IkeGenerateCookie (VOID)

UINT8 * IkeGenerateNonce (IN UINTN NonceSize)

VOID IkeHdrNetToHost (IN OUT IKE_HEADER *Heade

VOID IkeHdrHostToNet (IN OUT IKE_HEADER *Heade

IKE_PAYLOAD * IkePayloadAlloc (VOID)

VOID IkePayloadFree (IN IKE_PAYLOAD *IkePayload)

UINT32 IkeGenerateSpi (VOID)

EFI_STATUS IkeGenerateIv (IN UINT8 *IvBuffer, IN UINTN Iv

UINT8 IkeGetVersionFromSession (IN UINT8 *Session)

IPSEC_SPD_ENTRY * IkeSearchSpdEntry (IN EFI_IPSEC_SPD_SELECT

Variables

MODP_GROUP OakleyModpGroup []

IKE_ALG_GUID_INFO mIPsecEncrAlgInfo []

IKE_ALG_GUID_INFO mIPsecAuthAlgInfo []

Detailed Description

Common operation of the IKE.

Copyright (c) 2010 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by IkeDispatch(), Ikev2MatchSpdEntry(), and Ikev2SendIkePacket().

#define IKE_DEFAULT_TIMEOUT_INTERVAL 10000

Referenced by Ikev2SendIkePacket().

#define IKE_MAX_RETRY 4

Referenced by Ikev2ResendNotify().

#define IKE_NONCE_SIZE 16

Referenced by Ikev2InitPskGenerator(), and Ikev2InitPskParser().

#define IKE_PACKET_APPEND_PAYLOAD (IkePacket, IkePayload)

Value:

```
do {                                                                    \
    InsertTailList(&(IkePacket)->PayloadList, &(IkePayload)->ByPacket); \
  } while (0)
```

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), Ikev2DecodePacket(), Ikev2EncryptPacket(), Ikev2InfoGenerator(), Ikev2InitCertGenerator(), and Ikev2InitPskGenerator().

#define IKE_PACKET_END_PAYLOAD (IkePacket, Node)

Value:

```
Node = GetFirstNode (&(IkePacket)->PayloadList);                      \
  while (!IsNodeAtEnd (&(IkePacket)->PayloadList, Node)) {             \
    Node = GetNextNode (&(IkePacket)->PayloadList, Node);              \
  }                                                                     \
```

Referenced by Ikev2InitCertGenerator().

#define IKE_PACKET_REMOVE_PAYLOAD (IkePacket, IkePayload)

Value:

```
do {                                                                    \
    RemoveEntryList(&(IkePayload)->ByPacket);                            \
  } while (0)
```

Referenced by ClearAllPayloads().

#define IKE_PAYLOAD_BY_PACKET (a) CR(a,IKE_PAYLOAD,ByPacket,IKE_PAYLOAD_SIGNATURE)

Referenced by ClearAllPayloads(), IkeNetbufFromPacket(), IkePacketFree(), Ikev2AuthCertParser(), Ikev2AuthPskParser(), Ikev2ChildExchangeRequestType(), Ikev2EncodePacket(), Ikev2EncryptPacket(), Ikev2InfoParser(), Ikev2InitCertGenerator(), and Ikev2InitPskParser().

#define IKE_PAYLOAD_SIGNATURE SIGNATURE_32('I','K','E','P')

Referenced by IkePayloadAlloc().

#define IKE_SPI_BASE 0x10000

Function Documentation

UINT64 IkeGenerateCookie (VOID)

Call Crypto Lib to generate a random value with eight-octet length.

Returns:

the 64 byte vaule.

References IpSecCryptoIoGenerateRandomBytes().

Referenced by Ikev2InitPskParser(), and Ikev2SaSessionAlloc().

EFI_STATUS IkeGenerateIv (IN UINT8 * *IvBuffer*,
IN UINTN *IvSize*
)

Generate a random data for IV

Parameters:

[in] *IvBuffer* The pointer of the IV buffer.
[in] *IvSize* The IV size.

Return values:

EFI_SUCCESS Create a random data for IV.
otherwise Failed.

References IpSecCryptoIoGenerateRandomBytes().

Referenced by Ikev2EncryptPacket().

UINT8* IkeGenerateNonce (IN UINTN *NonceSize*)

Generate the random data for Nonce payload.

Parameters:

[in] *NonceSize* Size of the data in bytes.

Returns:
Buffer which contains the random data of the spcified size.

References IpSecCryptoIoGenerateRandomBytes().

Referenced by Ikev2InitPskGenerator(), and Ikev2InitPskParser().

UINT32 IkeGenerateSpi (VOID)
Generate an unused SPI

Returns:
a SPI in 4 bytes.

Generate an new SPI.

Returns:
a SPI in 4 bytes.

References mNextSpi.

Referenced by Ikev2ChildSaSessionAlloc().

UINT8 IkeGetVersionFromSession (IN UINT8 * *Session*)
Get the IKE Version from the IKE_SA_SESSION.

Parameters:
`[in]` *Session* Pointer of the IKE_SA_SESSION.

References IKEV2_SA_SESSION_SIGNATURE.

Referenced by IkeDeleteAllSas(), and IkeNegotiate().

VOID IkeHdrHostToNet (IN OUT IKE_HEADER * *Header*)
Convert the IKE Header from Host order to Network order.

Parameters:
`[in,out]` *Header* The pointer of the IKE_HEADER.

Referenced by Ikev2DecryptPacket(), Ikev2EncodePacket(), and Ikev2EncryptPacket().

VOID IkeHdrNetToHost (IN OUT IKE_HEADER * *Header*)
Convert the IKE Header from Network order to Host order.

Parameters:
`[in,out]` *Header* The pointer of the IKE_HEADER.

Referenced by IkePacketFromNetbuf(), and Ikev2DecodePacket().

IKE_PAYLOAD* IkePayloadAlloc (VOID)
Allocate a buffer of IKE_PAYLOAD and set its Signature.

Returns:
A buffer of IKE_PAYLOAD.

References IKE_PAYLOAD_SIGNATURE, and IKE_PAYLOAD::Signature.

Referenced by Ikev2CertGenerateAuthPayload(), Ikev2DecodePacket(), Ikev2EncryptPacket(), Ikev2GenerateCertIdPayload(), Ikev2GenerateCertificatePayload(), Ikev2GenerateCpPayload(), Ikev2GenerateDeletePayload(), Ikev2GenerateIdPayload(), Ikev2GenerateKePayload(), Ikev2GenerateNoncePayload(), Ikev2GenerateNotifyPayload(), Ikev2GenerateSaPayload(), Ikev2GenerateTsPayload(), and Ikev2PskGenerateAuthPayload().

VOID IkePayloadFree (IN IKE_PAYLOAD * *IkePayload*)
Free a specified IKE_PAYLOAD buffer.

Parameters:
`[in]` *IkePayload* Pointer of IKE_PAYLOAD to be freed.

Referenced by ClearAllPayloads(), IkePacketFree(), Ikev2AuthCertParser(), Ikev2CertGenerateAuthPayload(), Ikev2GenerateTsPayload(), Ikev2InitPskGenerator(), and Ikev2PskGenerateAuthPayload().

IPSEC_SPD_ENTRY* IkeSearchSpdEntry (IN EFI_IPSEC_SPD_SELECTOR * *SpdSel*)
Find SPD entry by a specified SPD selector.

Parameters:
`[in]` *SpdSel* Point to SPD Selector to be searched for.

Return values:
Point to Spd Entry if the SPD entry found.
NULL if not found.

Find SPD entry by a specified SPD selector.

Parameters:
`[in]` *SpdSel* Point to SPD Selector to be searched for.

Return values:
Point to SPD Entry if the SPD entry found.
NULL if not found.

References CompareSpdSelector(), IPSEC_SPD_ENTRY_FROM_LIST, mConfigData, and _IPSEC_SPD_ENTRY::Selector.

Referenced by Ikev2MatchSpdEntry().

Variable Documentation

IKE_ALG_GUID_INFO mIPsecAuthAlgInfo[]

IKE_ALG_GUID_INFO mIPsecEncrAlgInfo[]

MODP_GROUP OakleyModpGroup[]

Referenced by Ikev2GenerateSaDhPublicKey().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/IkePacket.c File Reference

Functions

IKE_PACKET * IkePacketAlloc (VOID)
VOID IkePacketFree (IN IKE_PACKET *IkePacket)
VOID EFIAPI IkePacketNetbufFree (IN VOID *Arg)
IKE_PACKET * IkePacketFromNetbuf (IN NET_BUF *Netbuf)
NET_BUF * IkeNetbufFromPacket (IN UINT8 *SessionCommon, IN IKE_PACKET *IkePacket, IN UINTN IkeType)

Detailed Description

IKE Packet related operation.

Copyright (c) 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
NET_BUF* IkeNetbufFromPacket ( IN UINT8 *          SessionCommon,
                               IN IKE_PACKET *     IkePacket,
                               IN UINTN            IkeType
                             )
```

Convert the format from IKE_PACKET to NetBuf.

Parameters:

[in]	*SessionCommon*	Pointer of related IKE_COMMON_SESSION
[in]	*IkePacket*	Pointer of IKE_PACKET to be copy to NetBuf
[in]	*IkeType*	The IKE type to pointer the packet is for which IKE phase. Now it supports IKE_SA_TYPE, IKE_CHILDSA_TYPE, IKE_INFO_TYPE.

Returns:

a pointer of Netbuff which contains the IKE_PACKE in network order.

References IKE_PAYLOAD_BY_PACKET, IkePacketNetbufFree(), Ikev2EncodePacket(), IKE_PAYLOAD::PayloadBuf, and IKE_PAYLOAD::PayloadSize.

Referenced by Ikev2SendIkePacket().

IKE_PACKET* IkePacketAlloc (VOID)
Allocate a buffer for the IKE_PACKET and intitalize its Header and payloadlist.

Returns:
The pointer of the IKE_PACKET.

References IKE_PACKET::Header, IKE_PACKET::PayloadList, and IKE_PACKET::RefCount.

Referenced by IkePacketFromNetbuf(), Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), Ikev2InfoGenerator(), and Ikev2InitPskGenerator().

VOID IkePacketFree (IN IKE_PACKET * *IkePacket*)
Free the IkePacket by the specified IKE_PACKET pointer.

Parameters:
[in] *IkePacket* The pointer of the IKE_PACKET to be freed.

References IKE_PAYLOAD_BY_PACKET, and IkePayloadFree().

Referenced by IkeDispatch(), IkePacketFromNetbuf(), Ikev2HandleChildSa(), Ikev2InitPskGenerator(), Ikev2NegotiateChildSa(), Ikev2NegotiateInfo(), Ikev2NegotiateSa(), Ikev2OnPacketSent(), Ikev2SaSessionCommonFree(), Ikev2SendIkePacket(), and Ikev2SessionCommonRefresh().

IKE_PACKET* IkePacketFromNetbuf (IN NET_BUF * *Netbuf*)
Copy the NetBuf into a IKE_PACKET sturcture.

Create a IKE_PACKET and fill the received IKE header into the header of IKE_PACKET and copy the recieved packet without IKE HEADER to the PayloadBuf of IKE_PACKET.

Parameters:
[in] *Netbuf* The pointer of the Netbuf which contains the whole received IKE packet.

Returns:
The pointer of the IKE_PACKET which contains the received packet.

References IKE_PACKET::Header, IkeHdrNetToHost(), IkePacketAlloc(), IkePacketFree(), IKE_HEADER::Length, IKE_PACKET::PayloadsBuf, and IKE_PACKET::PayloadTotalSize.

Referenced by IkeDispatch().

VOID EFIAPI IkePacketNetbufFree (IN VOID * *Arg*)
Callback funtion of NetbufFromExt()

Parameters:
[in] *Arg* The data passed from the NetBufFromExe().

Referenced by IkeNetbufFromPacket().

- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/IkePacket.h File Reference

Defines

"memItemRight" valign="bottom">IKE_PACKET_REF(p) ((p)->RefCount++)

Functions

IKE_PACKET *	IkePacketAlloc (VOID)
VOID	IkePacketFree (IN IKE_PACKET *IkePacket)
IKE_PACKET *	IkePacketFromNetbuf (IN NET_BUF *Netbuf)
NET_BUF *	IkeNetbufFromPacket (IN UINT8 *SessionCommon, IN IKE_PACKET *IkePacket, IN UINTN IkeType)

Detailed Description

IKE Packet related definitions and function declarations.

Copyright (c) 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"paramtype">p) ((p)->RefCount++)

Referenced by Ikev2SendIkePacket().

Function Documentation

```
NET_BUF* IkeNetbufFromPacket ( IN UINT8 *          SessionCommon,
                               IN IKE_PACKET *     IkePacket,
                               IN UINTN            IkeType
                             )
```

Convert the format from IKE_PACKET to NetBuf.

Parameters:

[in]	*SessionCommon*	Pointer of related IKE_COMMON_SESSION
[in]	*IkePacket*	Pointer of IKE_PACKET to be copy to NetBuf
[in]	*IkeType*	The IKE type to pointer the packet is for which IKE phase. Now it supports IKE_SA_TYPE, IKE_CHILDSA_TYPE, IKE_INFO_TYPE.

Returns:

A pointer of Netbuff which contains the contents of the IKE_PACKE in network order.

Convert the format from IKE_PACKET to NetBuf.

Parameters:

[in]	*SessionCommon*	Pointer of related IKE_COMMON_SESSION
[in]	*IkePacket*	Pointer of IKE_PACKET to be copy to NetBuf
[in]	*IkeType*	The IKE type to pointer the packet is for which IKE phase. Now it supports IKE_SA_TYPE, IKE_CHILDSA_TYPE, IKE_INFO_TYPE.

Returns:

a pointer of Netbuff which contains the IKE_PACKE in network order.

References IKE_PAYLOAD_BY_PACKET, IkePacketNetbufFree(), Ikev2EncodePacket(), IKE_PAYLOAD::PayloadBuf, and IKE_PAYLOAD::PayloadSize.

Referenced by Ikev2SendIkePacket().

IKE_PACKET* IkePacketAlloc (VOID)

Allocate a buffer for the IKE_PACKET and intitalize its Header and payloadlist.

Returns:

The pointer of the IKE_PACKET.

References IKE_PACKET::Header, IKE_PACKET::PayloadList, and IKE_PACKET::RefCount.

Referenced by IkePacketFromNetbuf(), Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), Ikev2InfoGenerator(), and Ikev2InitPskGenerator().

VOID IkePacketFree (IN IKE_PACKET * *IkePacket*)

Free the IkePacket by the specified IKE_PACKET pointer.

Parameters:

[in]	*IkePacket*	The pointer of the IKE_PACKET to be freed.

References IKE_PAYLOAD_BY_PACKET, and IkePayloadFree().

Referenced by IkeDispatch(), IkePacketFromNetbuf(), Ikev2HandleChildSa(), Ikev2InitPskGenerator(), Ikev2NegotiateChildSa(), Ikev2NegotiateInfo(), Ikev2NegotiateSa(), Ikev2OnPacketSent(), Ikev2SaSessionCommonFree(), Ikev2SendIkePacket(), and Ikev2SessionCommonRefresh().

IKE_PACKET* IkePacketFromNetbuf (IN NET_BUF * *Netbuf*)

Copy the NetBuf into a IKE_PACKET sturcture.

Create a IKE_PACKET and fill the received IKE header into the header of IKE_PACKET and copy the recieved packet without IKE HEADER to the PayloadBuf of IKE_PACKET.

Parameters:

`[in]` *Netbuf* The pointer of the Netbuf which contains the whole received IKE packet.

Returns:

The pointer of the IKE_PACKET which contains the received packet.

References IKE_PACKET::Header, IkeHdrNetToHost(), IkePacketAlloc(), IkePacketFree(), IKE_HEADER::Length, IKE_PACKET::PayloadsBuf, and IKE_PACKET::PayloadTotalSize.

Referenced by IkeDispatch().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/IkeService.c File Reference

Functions

IKE_UDP_SERVICE * IkeLookupUdp (IN IPSEC_PRIVATE_DATA *Private, IN EFI_HANDLE Handle, IN UINT8 IpVersion)
EFI_STATUS EFIAPI IkeConfigUdp4 (IN UDP_IO *UdpIo, IN VOID *Context)
EFI_STATUS EFIAPI IkeConfigUdp6 (IN UDP_IO *UdpIo, IN VOID *Context)
EFI_STATUS IkeOpenOutputUdp (IN IKE_UDP_SERVICE *UdpService, IN EFI_IP_ADDRESS *RemoteIp)
EFI_STATUS IkeOpenInputUdp4 (IN IPSEC_PRIVATE_DATA *Private, IN EFI_HANDLE Controller, IN EFI_HANDLE ImageHandle)
EFI_STATUS IkeOpenInputUdp6 (IN IPSEC_PRIVATE_DATA *Private, IN EFI_HANDLE Controller, IN EFI_HANDLE ImageHandle)
EFI_STATUS IkeNegotiate (IN IKE_UDP_SERVICE *UdpService, IN IPSEC_SPD_ENTRY *SpdEntry, IN EFI_IP_ADDRESS *RemoteIp)
VOID EFIAPI IkeDispatch (IN NET_BUF *Packet, IN UDP_END_POINT *EndPoint, IN EFI_STATUS IoStatus, IN VOID *Context)
VOID IkeDeleteAllSas (IN IPSEC_PRIVATE_DATA *Private, IN BOOLEAN IsDisableIpsec)

Variables

IKE_EXCHANGE_INTERFACE * mIkeExchange []
EFI_UDP4_CONFIG_DATA mUdp4Conf
EFI_UDP6_CONFIG_DATA mUdp6Conf

Detailed Description

Provide IPsec Key Exchange (IKE) service general interfaces.

Copyright (c) 2010 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI IkeConfigUdp4 (IN UDP_IO * *UdpIo*, IN VOID * *Context*)

Configure a UDPIO's UDP4 instance.

This fuction is called by the UdpIoCreateIo() to configures a UDP4 instance.

Parameters:

[in] *UdpIo* The UDP_IO to be configured.
[in] *Context* User-defined data when calling UdpIoCreateIo().

Return values:

EFI_SUCCESS The configuration succeeded.
Others The UDP4 instance fails to configure.

References mUdp4Conf.

Referenced by IkeOpenInputUdp4(), and IkeOpenOutputUdp().

```
EFI_STATUS EFIAPI IkeConfigUdp6 ( IN UDP_IO * UdpIo,
                                  IN VOID * Context
                                )
```

Configure a UDPIO's UDP6 instance.

This fuction is called by the UdpIoCreateIo()to configure a UDP6 instance.

Parameters:

[in] *UdpIo* The UDP_IO to be configured.
[in] *Context* User-defined data when calling UdpIoCreateIo().

Return values:

EFI_SUCCESS The configuration succeeded.
Others The configuration fails.

References mUdp6Conf.

Referenced by IkeOpenInputUdp6(), and IkeOpenOutputUdp().

```
VOID IkeDeleteAllSas ( IN IPSEC_PRIVATE_DATA * Private,
                       IN BOOLEAN              IsDisableIpsec
                     )
```

Delete all established IKE SAs and related Child SAs.

This function is the subfunction of the IpSecCleanupAllSa(). It first calls IkeDeleteChildSa() to delete all Child SAs then send out the related Information packet.

Parameters:

[in] *Private* Pointer of the IPSEC_PRIVATE_DATA
[in] *IsDisableIpsec* Indicate whether needs to disable IPsec.

References IkeGetVersionFromSession(), IkeStateSaDeleting, IKEV2_SA_SESSION_BY_SESSION, Ikev2SaSessionFree(), IPSEC_STATUS_DISABLED, IPSECCONFIG_STATUS_NAME, _IKE_EXCHANGE_INTERFACE::NegotiateInfo, IKEV2_SA_SESSION::SessionCommon, and IKEV2_SESSION_COMMON::State.

Referenced by IpSecCleanupAllSa().

```
VOID EFIAPI IkeDispatch ( IN NET_BUF *          Packet,
                          IN UDP_END_POINT *    EndPoint,
                          IN EFI_STATUS         IoStatus,
                          IN VOID *             Context
                        )
```

The generic interface when receive a IKE packet.

This function is called when UDP IO receives a IKE packet.

Parameters:

[in]	*Packet*	Point to received IKE packet.
[in]	*EndPoint*	Point to UDP_END_POINT which contains the information of Remote IP and Port.
[in]	*IoStatus*	The Status of Recieve Token.
[in]	*Context*	Point to data passed from the caller.

References IKE_HEADER::ExchangeType, _IKE_EXCHANGE_INTERFACE::HandleChildSa, _IKE_EXCHANGE_INTERFACE::HandleInfo, _IKE_EXCHANGE_INTERFACE::HandleSa, IKE_PACKET::Header, IKE_DEFAULT_PORT, IKE_MAJOR_VERSION, IKE_XCG_TYPE_AUTH, IKE_XCG_TYPE_CREATE_CHILD_SA, IKE_XCG_TYPE_IDENTITY_PROTECT, IKE_XCG_TYPE_INFO, IKE_XCG_TYPE_INFO2, IKE_XCG_TYPE_QM, IKE_XCG_TYPE_SA_INIT, IkeDispatch(), IkeOpenOutputUdp(), IkePacketFree(), IkePacketFromNetbuf(), IKE_UDP_SERVICE::Input, _IPSEC_PRIVATE_DATA::IpSec, IPSEC_PRIVATE_DATA_FROM_UDP4LIST, IPSEC_PRIVATE_DATA_FROM_UDP6LIST, IKE_UDP_SERVICE::IpVersion, IKE_UDP_SERVICE::ListHead, IKE_PACKET::RemotePeerIp, and IKE_HEADER::Version.

Referenced by IkeDispatch(), IkeOpenInputUdp4(), and IkeOpenInputUdp6().

```
IKE_UDP_SERVICE* IkeLookupUdp ( IN IPSEC_PRIVATE_DATA *  Private,
                                IN EFI_HANDLE            Handle,
                                IN UINT8                 IpVersion
                              )
```

Check if the NIC handle is binded to a Udp service.

Parameters:

[in]	*Private*	Pointer of IPSEC_PRIVATE_DATA.
[in]	*Handle*	The Handle of the NIC card.
[in]	*IpVersion*	The version of the IP stack.

Returns:

a pointer of IKE_UDP_SERVICE.

References IPSEC_UDP_SERVICE_FROM_LIST, and IKE_UDP_SERVICE::NicHandle.

Referenced by IkeOpenInputUdp4(), IkeOpenInputUdp6(), and IpSecLookupSadEntry().

```
EFI_STATUS IkeNegotiate ( IN IKE_UDP_SERVICE *   UdpService,
                          IN IPSEC_SPD_ENTRY *   SpdEntry,
                          IN EFI_IP_ADDRESS *    RemoteIp
                        )
```

The general interface of starting IPsec Key Exchange.

This function is called when a IKE negotiation to start getting a Key.

Parameters:

[in]	*UdpService*	Point to IKE_UDP_SERVICE which will be used for IKE packet sending.
[in]	*SpdEntry*	Point to the SPD entry related to the IKE negotiation.
[in]	*RemoteIp*	Point to EFI_IP_ADDRESS related to the IKE negotiation.

Return values:

EFI_SUCCESS	The Operation is successful.
EFI_ACCESS_DENIED	No related PAD entry was found.
EFI_INVALID_PARAMETER	The IKE version is not supported.

References _IPSEC_PAD_ENTRY::Data, IkeGetVersionFromSession(), IkeOpenOutputUdp(), _IPSEC_PRIVATE_DATA::Ikev2EstablishedList, Ikev2SaSessionLookup(), IPSEC_PRIVATE_DATA_FROM_UDP4LIST, IPSEC_PRIVATE_DATA_FROM_UDP6LIST, IpSecLookupPadEntry(), _IKE_EXCHANGE_INTERFACE::NegotiateChildSa, and _IKE_EXCHANGE_INTERFACE::NegotiateSa.

Referenced by IpSecLookupSadEntry().

```
EFI_STATUS IkeOpenInputUdp4 ( IN IPSEC_PRIVATE_DATA *   Private,
                              IN EFI_HANDLE             Controller,
                              IN EFI_HANDLE             ImageHandle
                            )
```

Open and configure a UDPIO of Udp4 for IKE packet receiving.

This function is called at the IPsecDriverBinding start. IPsec create a UDP4 and UDP4 IO for each NIC handle.

Parameters:

[in]	*Private*	Point to IPSEC_PRIVATE_DATA
[in]	*Controller*	Handler for NIC card.
[in]	*ImageHandle*	The handle that contains the EFI_DRIVER_BINDING_PROTOCOL instance.

Return values:

EFI_SUCCESS	The Operation is successful.
EFI_OUT_OF_RESOURCE	The required system resource can't be allocated.

References IKE_UDP_SERVICE::DefaultAddress, IkeConfigUdp4(), IkeDispatch(), IkeLookupUdp(), IKE_UDP_SERVICE::ImageHandle, IKE_UDP_SERVICE::Input, IKE_UDP_SERVICE::IpVersion, IKE_UDP_SERVICE::IsConfigured, IKE_UDP_SERVICE::List, IKE_UDP_SERVICE::ListHead, and IKE_UDP_SERVICE::NicHandle.

Referenced by IpSecStart().

```
EFI_STATUS IkeOpenInputUdp6 ( IN IPSEC_PRIVATE_DATA *   Private,
                              IN EFI_HANDLE             Controller,
                              IN EFI_HANDLE             ImageHandle
                            )
```

Open and configure a UDPIO of Udp6 for IKE packet receiving.

This function is called at the IPsecDriverBinding start. IPsec create a UDP6 and UDP6 IO for each NIC handle.

Parameters:

[in]	*Private*	Point to IPSEC_PRIVATE_DATA
[in]	*Controller*	Handler for NIC card.
[in]	*ImageHandle*	The handle that contains the EFI_DRIVER_BINDING_PROTOCOL instance.

Return values:

EFI_SUCCESS	The Operation is successful.
EFI_OUT_OF_RESOURCE	The required system resource can't be allocated.

References IKE_UDP_SERVICE::DefaultAddress, IkeConfigUdp6(), IkeDispatch(), IkeLookupUdp(), IKE_UDP_SERVICE::ImageHandle, IKE_UDP_SERVICE::Input, IKE_UDP_SERVICE::IpVersion, IKE_UDP_SERVICE::IsConfigured, IKE_UDP_SERVICE::List, IKE_UDP_SERVICE::ListHead, and IKE_UDP_SERVICE::NicHandle.

Referenced by IpSecStart().

```
EFI_STATUS IkeOpenOutputUdp ( IN IKE_UDP_SERVICE *   UdpService,
                              IN EFI_IP_ADDRESS *    RemoteIp
                            )
```

Open and configure the related output UDPIO for IKE packet sending.

If the UdpService is not configured, this fuction calls UdpIoCreatIo() to create UDPIO to bind this UdpService for IKE packet sending. If the UdpService has already been configured, then return.

Parameters:

[in]	*UdpService*	The UDP_IO to be configured.
[in]	*RemoteIp*	User-defined data when calling UdpIoCreateIo().

Return values:

EFI_SUCCESS	The configuration is successful.
Others	The configuration fails.

References IkeConfigUdp4(), and IkeConfigUdp6().

Referenced by IkeDispatch(), and IkeNegotiate().

Variable Documentation

IKE_EXCHANGE_INTERFACE* mIkeExchange[]

Initial value:

```
{
 &mIkev1Exchange,
 &mIkev2Exchange
}
```

EFI_UDP4_CONFIG_DATA mUdp4Conf

Initial value:

```
{
 FALSE,
 FALSE,
 FALSE,
 TRUE,



 0,
 64,
 FALSE,
 0,
 1000000,
 FALSE,
 {{0,0,0,0}},
 {{0,0,0,0}},
 IKE_DEFAULT_PORT,
 {{0,0,0,0}},
 0
}
```

Referenced by IkeConfigUdp4().

EFI_UDP6_CONFIG_DATA mUdp6Conf

Initial value:

```
{
 FALSE,
 FALSE,
 TRUE,



 0,
 128,
 0,
 1000000,

 {{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}},
 IKE_DEFAULT_PORT,
 {{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}},
 0
}
```

Referenced by IkeConfigUdp6().

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/IkeService.h File Reference

Data Structures

struct _IKE_EXCHANGE_INTERFA

Defines

"memItemRight" valign="bottom">IPSEC_CRYPTO_LIB_MEMORY 128 * 1024

Typedefs

typedef EFI_STATUS(* IKE_NEGOTIATE_SA)(IN IKE_UDP_SERVICE *UdpService, IN IPSEC_SPD_ENTRY *SpdEntr IN IPSEC_PAD_ENTRY *PadEntry, IN EFI_IP_ADDRES *RemoteIp)

typedef EFI_STATUS(* IKE_NEGOTIATE_CHILD_SA)(IN UINT8 *IkeSaSession, IN IPSEC_SPD_ENTRY *SpdEntr IN UINT8 *Context)

typedef EFI_STATUS(* IKE_NEGOTIATE_INFO)(IN UINT8 *IkeSaSession, IN UINT *Context)

typedef VOID(* IKE_HANDLE_SA)(IN IKE_UDP_SERVICE *UdpService, IN IKE_PACKET *IkePacket)

typedef VOID(* IKE_HANDLE_CHILD_SA)(I IKE_UDP_SERVICE *UdpService, IN IKE_PACKET *IkePacket)

typedef VOID(* IKE_HANDLE_INFO)(IN IKE_UDP_SERVICE *UdpService, IN IKE_PACKET *IkePacket)

typedef struct _IKE_EXCHANGE_INTERFACE IKE_EXCHANGE_INTERFAC

Functions

EFI_STATUS IkeOpenInputUdp4 (IN IPSEC_PRIVATE_DATA *Private, IN EFI_HANDLE Controller, IN EFI_HANDLE ImageHandle)

EFI_STATUS IkeOpenInputUdp6 (IN IPSEC_PRIVATE_DATA

*Private, IN EFI_HANDLE Controller, IN EFI_HANDLE ImageHandle)

EFI_STATUS IkeNegotiate (IN IKE_UDP_SERVICE *UdpService, IN IPSEC_SPD_ENTRY *SpdEntry, IN EFI_IP_ADDRESS *RemoteIp)

VOID EFIAPI IkeDispatch (IN NET_BUF *Packet, IN UDP_END_POINT *EndPoint, IN EFI_STATUS IoStatus, IN VOID *Context)

IKE_UDP_SERVICE * IkeLookupUdp (IN IPSEC_PRIVATE_DATA *Private, IN EFI_HANDLE Handle, IN UINT8 IpVersion)

VOID IkeDeleteAllSas (IN IPSEC_PRIVATE_DATA *Private, IN BOOLEAN IsDisableIpsec)

Variables

IKE_EXCHANGE_INTERFACE mIkev1Exchange

IKE_EXCHANGE_INTERFACE mIkev2Exchange

Detailed Description

Prototypes definitions of IKE service.

Copyright (c) 2010 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Typedef Documentation

typedef struct _IKE_EXCHANGE_INTERFACE IKE_EXCHANGE_INTERFACE

typedef VOID(* IKE_HANDLE_CHILD_SA)(IN IKE_UDP_SERVICE *UdpService, IN IKE_PACKET *IkePacket)

This is prototyp definition of the general interface when recived a IKE Packet xfor the Child SA establishing.

Parameters:

[in]	*UdpService*	Point to UDP service used to send IKE packet.
[in]	*IkePacket*	Point to received IKE packet.

typedef VOID(* IKE_HANDLE_INFO)(IN IKE_UDP_SERVICE *UdpService, IN IKE_PACKET *IkePacket)

This is prototype definition of the general interface when received a IKE information Packet.

Parameters:

[in]	*UdpService*	Point to UDP service used to send IKE packet.
[in]	*IkePacket*	Point to received IKE packet.

typedef VOID(* IKE_HANDLE_SA)(IN IKE_UDP_SERVICE *UdpService, IN IKE_PACKET *IkePacket)

This is prototype definition of the general interface when recived a IKE Pakcet for the IKE SA establishing.

Parameters:

[in]	*UdpService*	Point to UDP service used to send IKE Packet.
[in]	*IkePacket*	Point to received IKE packet.

typedef EFI_STATUS(* IKE_NEGOTIATE_CHILD_SA)(IN UINT8 *IkeSaSession, IN IPSEC_SPD_ENTRY *SpdEntry, IN UINT8 *Context)

This is prototype definition fo general interface to start a IKE negotiation at Quick Mode.

This function will be called when the related IKE SA is existed and start to create a Child SA.

Parameters:

[in]	*IkeSaSession*	Point to IKE SA Session related to this Negotiation.
[in]	*SpdEntry*	Point to SPD entry related to this Negotiation.
[in]	*Context*	Point to data passed from the caller.

Return values:

EFI_SUCCESS	The operation is successful.
Otherwise	The operation is failed.

typedef EFI_STATUS(* IKE_NEGOTIATE_INFO)(IN UINT8 *IkeSaSession, IN UINT8 *Context)

This is prototype definition of the general interface when initialize a Inforamtion Exchange.

Parameters:

[in]	*IkeSaSession*	Point to IKE SA Session related to.
[in]	*Context*	Point to data passed from caller.

typedef EFI_STATUS(* IKE_NEGOTIATE_SA)(IN IKE_UDP_SERVICE *UdpService, IN

IPSEC_SPD_ENTRY *SpdEntry, IN IPSEC_PAD_ENTRY *PadEntry, IN EFI_IP_ADDRESS *RemoteIp)
This is prototype definition of general interface to intialize a IKE negotiation.

Parameters:

[in]	*UdpService*	Point to Udp Servcie used for the IKE packet sending.
[in]	*SpdEntry*	Point to SPD entry related to this IKE negotiation.
[in]	*PadEntry*	Point to PAD entry related to this IKE negotiation.
[in]	*RemoteIp*	Point to IP Address which the remote peer to negnotiate.

Return values:

EFI_SUCCESS The operation is successful.

Returns:

Otherwise The operation is failed.

Function Documentation

```
VOID IkeDeleteAllSas ( IN IPSEC_PRIVATE_DATA *   Private,
                       IN BOOLEAN                IsDisableIpsec
                     )
```

Delete all established IKE SAs and related Child SAs.

This function is the subfunction of the IpSecCleanupAllSa(). It first calls IkeDeleteChildSa() to delete all Child SAs then send out the related Information packet.

Parameters:

[in]	*Private*	Pointer of the IPSEC_PRIVATE_DATA.
[in]	*IsDisableIpsec*	Indicate whether needs to disable IPsec.

Delete all established IKE SAs and related Child SAs.

This function is the subfunction of the IpSecCleanupAllSa(). It first calls IkeDeleteChildSa() to delete all Child SAs then send out the related Information packet.

Parameters:

[in]	*Private*	Pointer of the IPSEC_PRIVATE_DATA
[in]	*IsDisableIpsec*	Indicate whether needs to disable IPsec.

References IkeGetVersionFromSession(), IkeStateSaDeleting, IKEV2_SA_SESSION_BY_SESSION, Ikev2SaSessionFree(), IPSEC_STATUS_DISABLED, IPSECCONFIG_STATUS_NAME, _IKE_EXCHANGE_INTERFACE::NegotiateInfo, IKEV2_SA_SESSION::SessionCommon, and IKEV2_SESSION_COMMON::State.

Referenced by IpSecCleanupAllSa().

```
VOID EFIAPI IkeDispatch ( IN NET_BUF *          Packet,
                          IN UDP_END_POINT *    EndPoint,
                          IN EFI_STATUS         IoStatus,
                          IN VOID *             Context
```

)

The general interface when receive a IKE packet.

This function is called when UDP IO receives a IKE packet.

Parameters:

[in]	*Packet*	Point to received IKE packet.
[in]	*EndPoint*	Point to UDP_END_POINT which contains the information of Remote IP and Port.
[in]	*IoStatus*	The Status of Recieve Token.
[in]	*Context*	Point to data passed from the caller.

The generic interface when receive a IKE packet.

This function is called when UDP IO receives a IKE packet.

Parameters:

[in]	*Packet*	Point to received IKE packet.
[in]	*EndPoint*	Point to UDP_END_POINT which contains the information of Remote IP and Port.
[in]	*IoStatus*	The Status of Recieve Token.
[in]	*Context*	Point to data passed from the caller.

References IKE_HEADER::ExchangeType, _IKE_EXCHANGE_INTERFACE::HandleChildSa, _IKE_EXCHANGE_INTERFACE::HandleInfo, _IKE_EXCHANGE_INTERFACE::HandleSa, IKE_PACKET::Header, IKE_DEFAULT_PORT, IKE_MAJOR_VERSION, IKE_XCG_TYPE_AUTH, IKE_XCG_TYPE_CREATE_CHILD_SA, IKE_XCG_TYPE_IDENTITY_PROTECT, IKE_XCG_TYPE_INFO, IKE_XCG_TYPE_INFO2, IKE_XCG_TYPE_QM, IKE_XCG_TYPE_SA_INIT, IkeDispatch(), IkeOpenOutputUdp(), IkePacketFree(), IkePacketFromNetbuf(), IKE_UDP_SERVICE::Input, _IPSEC_PRIVATE_DATA::IpSec, IPSEC_PRIVATE_DATA_FROM_UDP4LIST, IPSEC_PRIVATE_DATA_FROM_UDP6LIST, IKE_UDP_SERVICE::IpVersion, IKE_UDP_SERVICE::ListHead, IKE_PACKET::RemotePeerIp, and IKE_HEADER::Version.

Referenced by IkeDispatch(), IkeOpenInputUdp4(), and IkeOpenInputUdp6().

```
IKE_UDP_SERVICE* IkeLookupUdp ( IN IPSEC_PRIVATE_DATA *  Private,
                                IN EFI_HANDLE            Handle,
                                IN UINT8                 IpVersion
                              )
```

Check if the NIC handle is binded to a Udp service.

Parameters:

[in]	*Private*	Pointer of IPSEC_PRIVATE_DATA
[in]	*Handle*	The Handle of the NIC card
[in]	*IpVersion*	The version of the IP stack.

Returns:

a pointer of IKE_UDP_SERVICE.

Check if the NIC handle is binded to a Udp service.

Parameters:

[in]	*Private*	Pointer of IPSEC_PRIVATE_DATA.

[in] *Handle* The Handle of the NIC card.
[in] *IpVersion* The version of the IP stack.

Returns:
a pointer of IKE_UDP_SERVICE.

References IPSEC_UDP_SERVICE_FROM_LIST, and IKE_UDP_SERVICE::NicHandle.

Referenced by IkeOpenInputUdp4(), IkeOpenInputUdp6(), and IpSecLookupSadEntry().

```
EFI_STATUS IkeNegotiate ( IN IKE_UDP_SERVICE *   UdpService,
                          IN IPSEC_SPD_ENTRY *   SpdEntry,
                          IN EFI_IP_ADDRESS *    RemoteIp
                        )
```

The general interface of starting IPsec Key Exchange.

This function is called when start a IKE negotiation to get a Key.

Parameters:
[in] *UdpService* Point to IKE_UDP_SERVICE which will be used for IKE packet sending.
[in] *SpdEntry* Point to the SPD entry related to the IKE negotiation.
[in] *RemoteIp* Point to EFI_IP_ADDRESS related to the IKE negotiation.

Return values:
EFI_SUCCESS The Operation is successful.
EFI_ACCESS_DENIED No related PAD entry was found.

The general interface of starting IPsec Key Exchange.

This function is called when a IKE negotiation to start getting a Key.

Parameters:
[in] *UdpService* Point to IKE_UDP_SERVICE which will be used for IKE packet sending.
[in] *SpdEntry* Point to the SPD entry related to the IKE negotiation.
[in] *RemoteIp* Point to EFI_IP_ADDRESS related to the IKE negotiation.

Return values:
EFI_SUCCESS The Operation is successful.
EFI_ACCESS_DENIED No related PAD entry was found.
EFI_INVALID_PARAMETER The IKE version is not supported.

References _IPSEC_PAD_ENTRY::Data, IkeGetVersionFromSession(), IkeOpenOutputUdp(), _IPSEC_PRIVATE_DATA::Ikev2EstablishedList, Ikev2SaSessionLookup(), IPSEC_PRIVATE_DATA_FROM_UDP4LIST, IPSEC_PRIVATE_DATA_FROM_UDP6LIST, IpSecLookupPadEntry(), _IKE_EXCHANGE_INTERFACE::NegotiateChildSa, and _IKE_EXCHANGE_INTERFACE::NegotiateSa.

Referenced by IpSecLookupSadEntry().

```
EFI_STATUS IkeOpenInputUdp4 ( IN IPSEC_PRIVATE_DATA *   Private,
```

```
                                IN EFI_HANDLE          Controller,
                                IN EFI_HANDLE          ImageHandle
                              )
```

Open and configure a UDPIO of Udp4 for IKE packet receiving.

This function is called at the IPsecDriverBinding start. IPsec create a UDP4 and a UDP4 IO for each NIC handle.

Parameters:

[in]	*Private*	Point to IPSEC_PRIVATE_DATA
[in]	*Controller*	Handler for NIC card.
[in]	*ImageHandle*	The handle that contains the EFI_DRIVER_BINDING_PROTOCOL instance.

Return values:

EFI_SUCCESS	The Operation is successful.
EFI_OUT_OF_RESOURCE	The required system resource can't be allocated.

Open and configure a UDPIO of Udp4 for IKE packet receiving.

This function is called at the IPsecDriverBinding start. IPsec create a UDP4 and UDP4 IO for each NIC handle.

Parameters:

[in]	*Private*	Point to IPSEC_PRIVATE_DATA
[in]	*Controller*	Handler for NIC card.
[in]	*ImageHandle*	The handle that contains the EFI_DRIVER_BINDING_PROTOCOL instance.

Return values:

EFI_SUCCESS	The Operation is successful.
EFI_OUT_OF_RESOURCE	The required system resource can't be allocated.

References IKE_UDP_SERVICE::DefaultAddress, IkeConfigUdp4(), IkeDispatch(), IkeLookupUdp(), IKE_UDP_SERVICE::ImageHandle, IKE_UDP_SERVICE::Input, IKE_UDP_SERVICE::IpVersion, IKE_UDP_SERVICE::IsConfigured, IKE_UDP_SERVICE::List, IKE_UDP_SERVICE::ListHead, and IKE_UDP_SERVICE::NicHandle.

Referenced by IpSecStart().

```
EFI_STATUS IkeOpenInputUdp6 ( IN IPSEC_PRIVATE_DATA *  Private,
                              IN EFI_HANDLE            Controller,
                              IN EFI_HANDLE            ImageHandle
                            )
```

Open and configure a UDPIO of Udp6 for IKE packet receiving.

This function is called at the IPsecDriverBinding start. IPsec create a UDP6 and UDP6 IO for each NIC handle.

Parameters:

[in]	*Private*	Point to IPSEC_PRIVATE_DATA
[in]	*Controller*	Handler for NIC card.
[in]	*ImageHandle*	The handle that contains the EFI_DRIVER_BINDING_PROTOCOL instance.

Return values:

EFI_SUCCESS	The Operation is successful.
EFI_OUT_OF_RESOURCE	The required system resource can't be allocated.

References IKE_UDP_SERVICE::DefaultAddress, IkeConfigUdp6(), IkeDispatch(), IkeLookupUdp(), IKE_UDP_SERVICE::ImageHandle, IKE_UDP_SERVICE::Input, IKE_UDP_SERVICE::IpVersion, IKE_UDP_SERVICE::IsConfigured, IKE_UDP_SERVICE::List, IKE_UDP_SERVICE::ListHead, and IKE_UDP_SERVICE::NicHandle.

Referenced by IpSecStart().

Variable Documentation

IKE_EXCHANGE_INTERFACE mIkev1Exchange

IKE_EXCHANGE_INTERFACE mIkev2Exchange

Referenced by Ikev2LifetimeNotify().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/IpSecConfigImpl.c File Reference

Functions

EFI_STATUS IpSecConfigRestore (IN IPSEC_PRIVATE_DATA *Private)
BOOLEAN IsInAddressInfoList (IN EFI_IP_ADDRESS_INFO *AddressInfo, IN EFI_IP_ADDRESS_INFO *AddressInfoList, IN UINT32 AddressCount)
BOOLEAN CompareSpdSelector (IN EFI_IPSEC_CONFIG_SELECTOR *Selector1, IN EFI_IPSEC_CONFIG_SELECTOR *Selector2)
BOOLEAN IsSubSpdSelector (IN EFI_IPSEC_CONFIG_SELECTOR *Selector1, IN EFI_IPSEC_CONFIG_SELECTOR *Selector2)
BOOLEAN CompareSaId (IN EFI_IPSEC_CONFIG_SELECTOR *Selector1, IN EFI_IPSEC_CONFIG_SELECTOR *Selector2)
BOOLEAN ComparePadId (IN EFI_IPSEC_CONFIG_SELECTOR *Selector1, IN EFI_IPSEC_CONFIG_SELECTOR *Selector2)
BOOLEAN IsZeroSpdSelector (IN EFI_IPSEC_CONFIG_SELECTOR *Selector)
BOOLEAN IsZeroSaId (IN EFI_IPSEC_CONFIG_SELECTOR *Selector)
BOOLEAN IsZeroPadId (IN EFI_IPSEC_CONFIG_SELECTOR *Selector)
EFI_STATUS DuplicateSpdSelector (IN OUT EFI_IPSEC_CONFIG_SELECTOR *DstSel, IN EFI_IPSEC_CONFIG_SELECTOR *SrcSel, IN OUT UINTN *Size)
EFI_STATUS DuplicateSaId (IN OUT EFI_IPSEC_CONFIG_SELECTOR *DstSel, IN EFI_IPSEC_CONFIG_SELECTOR *SrcSel, IN OUT UINTN *Size)
EFI_STATUS DuplicatePadId (IN OUT EFI_IPSEC_CONFIG_SELECTOR *DstSel, IN EFI_IPSEC_CONFIG_SELECTOR *SrcSel, IN OUT UINTN *Size)
VOID FixSpdEntry (IN EFI_IPSEC_SPD_SELECTOR *Selector, IN OUT EFI_IPSEC_SPD_DATA *Data)
VOID FixSadEntry (IN EFI_IPSEC_SA_ID *SaId, IN OUT EFI_IPSEC_SA_DATA2 *Data)
VOID FixPadEntry (IN EFI_IPSEC_PAD_ID *PadId, IN OUT EFI_IPSEC_PAD_DATA *Data)
VOID UnfixSpdEntry (IN OUT EFI_IPSEC_SPD_SELECTOR *Selector, IN OUT EFI_IPSEC_SPD_DATA *Data)
VOID UnfixSadEntry (IN OUT EFI_IPSEC_SA_ID *SaId, IN OUT EFI_IPSEC_SA_DATA2 *Data)
VOID UnfixPadEntry (IN EFI_IPSEC_PAD_ID *PadId, IN OUT EFI_IPSEC_PAD_DATA *Data)
EFI_STATUS SetSpdEntry (IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data, IN VOID *Context)
EFI_STATUS SetSadEntry (IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data, IN VOID *Context)
EFI_STATUS SetPadEntry (IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data, IN VOID *Context)
EFI_STATUS GetSpdEntry (IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN OUT UINTN *DataSize, OUT VOID *Data)
EFI_STATUS GetSadEntry (IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN OUT UINTN *DataSize, OUT VOID *Data)
EFI_STATUS GetPadEntry (IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN OUT UINTN *DataSize, OUT VOID *Data)

VOID IpSecDuplicateProcessPolicy (IN EFI_IPSEC_PROCESS_POLICY *Dst, IN EFI_IPSEC_PROCESS_POLICY *Src)
UINTN IpSecGetSizeOfEfiSpdData (IN EFI_IPSEC_SPD_DATA *SpdData)
UINTN IpSecGetSizeOfSpdData (IN IPSEC_SPD_DATA *SpdData)
EFI_STATUS IpSecGetVariable (IN CHAR16 *VariableName, IN EFI_GUID *VendorGuid, IN UINT32 *Attributes, IN OUT UINTN *DataSize, IN VOID *Data)
EFI_STATUS IpSecSetVariable (IN CHAR16 *VariableName, IN EFI_GUID *VendorGuid, IN UINT32 Attributes, IN UINTN DataSize, IN VOID *Data)
EFI_STATUS EFIAPI EfiIpSecConfigGetData (IN EFI_IPSEC_CONFIG_PROTOCOL *This, IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN OUT UINTN *DataSize, OUT VOID *Data)
EFI_STATUS EFIAPI EfiIpSecConfigSetData (IN EFI_IPSEC_CONFIG_PROTOCOL *This, IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data, IN EFI_IPSEC_CONFIG_SELECTOR *InsertBefore)
EFI_STATUS EFIAPI EfiIpSecConfigGetNextSelector (IN EFI_IPSEC_CONFIG_PROTOCOL *This, IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN OUT UINTN *SelectorSize, IN OUT EFI_IPSEC_CONFIG_SELECTOR *Selector)
EFI_STATUS EFIAPI EfiIpSecConfigRegisterNotify (IN EFI_IPSEC_CONFIG_PROTOCOL *This, IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN EFI_EVENT Event)
EFI_STATUS EFIAPI EfiIpSecConfigUnregisterNotify (IN EFI_IPSEC_CONFIG_PROTOCOL *This, IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN EFI_EVENT Event)
EFI_STATUS IpSecCopyPolicyEntry (IN EFI_IPSEC_CONFIG_DATA_TYPE Type, IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data, IN UINTN SelectorSize, IN UINTN DataSize, IN OUT IPSEC_VARIABLE_BUFFER *Buffer)
EFI_STATUS IpSecVisitConfigData (IN EFI_IPSEC_CONFIG_DATA_TYPE DataType, IN IPSEC_COPY_POLICY_ENTRY Routine, IN VOID *Context)
EFI_STATUS IpSecConfigSave (VOID)
EFI_STATUS IpSecConfigInitialize (IN OUT IPSEC_PRIVATE_DATA *Private)

Variables

LIST_ENTRY mConfigData [IPsecConfigDataTypeMaximum]
BOOLEAN mSetBySelf = FALSE
IPSEC_COMPARE_SELECTOR mCompareSelector []
IPSEC_IS_ZERO_SELECTOR mIsZeroSelector []
IPSEC_DUPLICATE_SELECTOR mDuplicateSelector []
IPSEC_FIX_POLICY_ENTRY mFixPolicyEntry []
IPSEC_FIX_POLICY_ENTRY mUnfixPolicyEntry []
IPSEC_SET_POLICY_ENTRY mSetPolicyEntry []
IPSEC_GET_POLICY_ENTRY mGetPolicyEntry []
EFI_IPSEC_CONFIG_PROTOCOL mIpSecConfigInstance

Detailed Description

The implementation of IPSEC_CONFIG_PROTOCOL.

Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
BOOLEAN ComparePadId ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector1,
                       IN EFI_IPSEC_CONFIG_SELECTOR *  Selector2
                     )
```

Compare two PAD IDs.

Parameters:

- [in] *Selector1* Pointer of first PAD ID.
- [in] *Selector2* Pointer of second PAD ID.

Return values:

- *TRUE* This two Selectors have the same PAD ID.
- *FALSE* This two Selecotrs don't have the same PAD ID.

Referenced by GetPadEntry(), and SetPadEntry().

```
BOOLEAN CompareSaId ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector1,
                      IN EFI_IPSEC_CONFIG_SELECTOR *  Selector2
                    )
```

Compare two SA IDs.

Parameters:

- [in] *Selector1* Pointer of first SA ID.
- [in] *Selector2* Pointer of second SA ID.

Return values:

- *TRUE* This two Selectors have the same SA ID.
- *FALSE* This two Selecotrs don't have the same SA ID.

Referenced by GetSadEntry(), SetSadEntry(), and SetSpdEntry().

```
BOOLEAN CompareSpdSelector ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector1,
                             IN EFI_IPSEC_CONFIG_SELECTOR *  Selector2
                           )
```

Compare two SPD Selectors.

Compare two SPD Selector by the fields of LocalAddressCount/RemoteAddressCount/NextLayerProtocol/LocalPort/LocalPortRange/RemotePort/RemotePortRange and the Local Addresses and remote Addresses.

Parameters:

`[in]` *Selector1* Pointer of first SPD Selector.
`[in]` *Selector2* Pointer of second SPD Selector.

Return values:

TRUE This two Selector have the same value in above fields.
FALSE Not all above fields have the same value in these two Selectors.

References IsInAddressInfoList().

Referenced by GetSpdEntry(), IkeSearchSpdEntry(), and SetSpdEntry().

```
EFI_STATUS DuplicatePadId ( IN OUT EFI_IPSEC_CONFIG_SELECTOR *   DstSel,
                            IN EFI_IPSEC_CONFIG_SELECTOR *       SrcSel,
                            IN OUT UINTN *                       Size
                          )
```

Copy Source PAD ID to the Destination PAD ID.

Parameters:

`[in,out]` *DstSel* Pointer of Destination PAD ID.
`[in]` *SrcSel* Pointer of Source PAD ID.
`[in,out]` *Size* The size of the Destination PAD ID. If it not NULL and its value less than the size of Source PAD ID, the value of Source PAD ID's size will be passed to caller by this parameter.

Return values:

EFI_INVALID_PARAMETER If the Destination or Source PAD ID is NULL.
EFI_BUFFER_TOO_SMALL If the input Size less than size of source PAD ID .
EFI_SUCCESS Copy Source PAD ID to the Destination PAD ID successfully.

```
EFI_STATUS DuplicateSaId ( IN OUT EFI_IPSEC_CONFIG_SELECTOR *   DstSel,
                           IN EFI_IPSEC_CONFIG_SELECTOR *       SrcSel,
                           IN OUT UINTN *                       Size
                         )
```

Copy Source SA ID to the Destination SA ID.

Parameters:

`[in,out]` *DstSel* Pointer of Destination SA ID.
`[in]` *SrcSel* Pointer of Source SA ID.
`[in,out]` *Size* The size of the Destination SA ID. If it not NULL and its value less than the size of Source SA ID, the value of Source SA ID's size will be passed to caller by this parameter.

Return values:

EFI_INVALID_PARAMETER	If the Destination or Source SA ID is NULL.
EFI_BUFFER_TOO_SMALL	If the input Size less than size of source SA ID.
EFI_SUCCESS	Copy Source SA ID to the Destination SA ID successfully.

```
EFI_STATUS DuplicateSpdSelector ( IN OUT EFI_IPSEC_CONFIG_SELECTOR *   DstSel,
                                  IN EFI_IPSEC_CONFIG_SELECTOR *       SrcSel,
                                  IN OUT UINTN *                       Size
                                )
```

Copy Source SPD Selector to the Destination SPD Selector.

Parameters:

`[in,out]`	*DstSel*	Pointer of Destination SPD Selector.
`[in]`	*SrcSel*	Pointer of Source SPD Selector.
`[in,out]`	*Size*	The size of the Destination SPD Selector. If it not NULL and its value less than the size of Source SPD Selector, the value of Source SPD Selector's size will be passed to caller by this parameter.

Return values:

EFI_INVALID_PARAMETER	If the Destination or Source SPD Selector is NULL
EFI_BUFFER_TOO_SMALL	If the input Size is less than size of the Source SPD Selector.
EFI_SUCCESS	Copy Source SPD Selector to the Destination SPD Selector successfully.

References SIZE_OF_SPD_SELECTOR.

Referenced by GetSadEntry(), SetSadEntry(), and SetSpdEntry().

```
EFI_STATUS EFIAPI EfiIpSecConfigGetData ( IN EFI_IPSEC_CONFIG_PROTOCOL *   This,
                                          IN EFI_IPSEC_CONFIG_DATA_TYPE    DataType,
                                          IN EFI_IPSEC_CONFIG_SELECTOR *   Selector,
                                          IN OUT UINTN *                   DataSize,
                                          OUT VOID *                       Data
                                        )
```

Return the configuration value for the EFI IPsec driver.

This function lookup the data entry from IPsec database or IKEv2 configuration information. The expected data type and unique identification are described in DataType and Selector parameters.

Parameters:

`[in]`	*This*	Pointer to the EFI_IPSEC_CONFIG_PROTOCOL instance.
`[in]`	*DataType*	The type of data to retrieve.
`[in]`	*Selector*	Pointer to an entry selector that is an identifier of the IPsec configuration data entry.
`[in,out]`	*DataSize*	On output the size of data returned in Data.
`[out]`	*Data*	The buffer to return the contents of the IPsec configuration data. The type of the data buffer associated with the DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.

EFI_INVALID_PARAMETER	One or more of the followings are TRUE: ◊ This is NULL. ◊ Selector is NULL. ◊ DataSize is NULL. ◊ Data is NULL and *DataSize is not zero
EFI_NOT_FOUND	The configuration data specified by Selector is not found.
EFI_UNSUPPORTED	The specified DataType is not supported.
EFI_BUFFER_TOO_SMALL	The DataSize is too small for the result. DataSize has been updated with the size needed to complete the request.

Referenced by IpSecVisitConfigData().

```
EFI_STATUS EFIAPI
EfiIpSecConfigGetNextSelector
                                ( IN EFI_IPSEC_CONFIG_PROTOCOL *     This,
                                  IN EFI_IPSEC_CONFIG_DATA_TYPE      DataType,
                                  IN OUT UINTN *                     SelectorSize,
                                  IN OUT
                                  EFI_IPSEC_CONFIG_SELECTOR *        Selector
                                )
```

Enumerates the current selector for IPsec configuration data entry.

This function is called multiple times to retrieve the entry Selector in IPsec configuration database. On each call to GetNextSelector(), the next entry Selector are retrieved into the output interface.

If the entire IPsec configuration database has been iterated, the error EFI_NOT_FOUND is returned. If the Selector buffer is too small for the next Selector copy, an EFI_BUFFER_TOO_SMALL error is returned, and SelectorSize is updated to reflect the size of buffer needed.

On the initial call to GetNextSelector() to start the IPsec configuration database search, a pointer to the buffer with all zero value is passed in Selector. Calls to SetData() between calls to GetNextSelector may produce unpredictable results.

Parameters:

`[in]`	*This*	Pointer to the EFI_IPSEC_CONFIG_PROTOCOL instance.
`[in]`	*DataType*	The type of IPsec configuration data to retrieve.
`[in,out]`	*SelectorSize*	The size of the Selector buffer.
`[in,out]`	*Selector*	On input, supplies the pointer to last Selector that was returned by GetNextSelector(). On output, returns one copy of the current entry Selector of a given DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_INVALID_PARAMETER	One or more of the followings are TRUE: ◊ This is NULL. ◊ SelectorSize is NULL. ◊ Selector is NULL.
EFI_NOT_FOUND	The next configuration data entry was not found.
EFI_UNSUPPORTED	The specified DataType is not supported.
EFI_BUFFER_TOO_SMALL	The SelectorSize is too small for the result. This parameter has been updated with the size needed to complete the search request.

References mCompareSelector, mConfigData, and IPSEC_COMMON_POLICY_ENTRY::Selector.

Referenced by Ikev2ChildSaSilentDelete(), and IpSecVisitConfigData().

```
EFI_STATUS EFIAPI
EfiIpSecConfigRegisterNotify
                                    ( IN EFI_IPSEC_CONFIG_PROTOCOL *  This,
                                      IN EFI_IPSEC_CONFIG_DATA_TYPE   DataType,
                                      IN EFI_EVENT                    Event
                                    )
```

Register an event that is to be signaled whenever a configuration process on the specified IPsec configuration information is done.

The register function is not surpport now and always returns EFI_UNSUPPORTED.

Parameters:

[in]	*This*	Pointer to the EFI_IPSEC_CONFIG_PROTOCOL instance.
[in]	*DataType*	The type of data to be registered the event for.
[in]	*Event*	The event to be registered.

Return values:

EFI_SUCCESS	The event is registered successfully.
EFI_INVALID_PARAMETER	This is NULL or Event is NULL.
EFI_ACCESS_DENIED	The Event is already registered for the DataType.
EFI_UNSUPPORTED	The notify registration is unsupported, or the specified DataType is not supported.

```
EFI_STATUS EFIAPI EfiIpSecConfigSetData  ( IN EFI_IPSEC_CONFIG_PROTOCOL *  This,
                                           IN EFI_IPSEC_CONFIG_DATA_TYPE   DataType,
                                           IN EFI_IPSEC_CONFIG_SELECTOR *  Selector,
                                           IN VOID *                       Data,
                                           IN EFI_IPSEC_CONFIG_SELECTOR *  InsertBefore
                                         )
```

Set the security association, security policy and peer authorization configuration information for the EFI IPsec driver.

This function is used to set the IPsec configuration information of type DataType for the EFI IPsec driver. The IPsec configuration data has a unique selector/identifier separately to identify a data entry. The selector structure depends on DataType's definition. Using SetData() with a Data of NULL causes the IPsec configuration data entry identified by DataType and Selector to be deleted.

Parameters:

[in]	*This*	Pointer to the EFI_IPSEC_CONFIG_PROTOCOL instance.
[in]	*DataType*	The type of data to be set.
[in]	*Selector*	Pointer to an entry selector on operated configuration data specified by DataType. A NULL Selector causes the entire specified-type configuration information to be flushed.
[in]	*Data*	The data buffer to be set. The structure of the data buffer is associated with the DataType.
[in]	*InsertBefore*	Pointer to one entry selector which describes the expected position the new data

entry will be added. If InsertBefore is NULL, the new entry will be appended to the end of the database.

Return values:

EFI_SUCCESS	The specified configuration entry data was set successfully.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: ◊ This is NULL.
EFI_UNSUPPORTED	The specified DataType is not supported.
EFI_OUT_OF_RESOURCED	The required system resource could not be allocated.

References IpSecConfigSave(), and mSetBySelf.

Referenced by Ikev2ChildSaSilentDelete(), Ikev2StoreSaData(), and IpSecConfigRestore().

```
EFI_STATUS EFIAPI
EfiIpSecConfigUnregisterNotify ( IN EFI_IPSEC_CONFIG_PROTOCOL *   This,
                                 IN EFI_IPSEC_CONFIG_DATA_TYPE    DataType,
                                 IN EFI_EVENT                     Event
                               )
```

Remove the specified event that was previously registered on the specified IPsec configuration data.

This function is not support now and alwasy return EFI_UNSUPPORTED.

Parameters:

`[in]`	*This*	Pointer to the EFI_IPSEC_CONFIG_PROTOCOL instance.
`[in]`	*DataType*	The configuration data type to remove the registered event for.
`[in]`	*Event*	The event to be unregistered.

Return values:

EFI_SUCCESS	The event was removed successfully.
EFI_NOT_FOUND	The Event specified by DataType could not be found in the database.
EFI_INVALID_PARAMETER	This is NULL or Event is NULL.
EFI_UNSUPPORTED	The notify registration is unsupported, or the specified DataType is not supported.

```
VOID FixPadEntry ( IN EFI_IPSEC_PAD_ID *          PadId,
                   IN OUT EFI_IPSEC_PAD_DATA *    Data
                 )
```

Fix the value of some members of PAD ID.

This function is called by IpSecCopyPolicyEntry()which copy the Policy Entry into the Variable. Since some members in PAD ID are pointers, a physical address to relative address conversion is required before copying this PAD into the variable.

Parameters:

`[in]`	*PadId*	Pointer of PAD ID.
`[in,out]`	*Data*	Pointer of PAD Data.

References FIX_REF_BUF_ADDR.

```
VOID FixSadEntry ( IN EFI_IPSEC_SA_ID *          SaId,
                   IN OUT EFI_IPSEC_SA_DATA2 *   Data
                 )
```

Fix the value of some members of SA ID.

This function is called by IpSecCopyPolicyEntry()which copy the Policy Entry into the Variable. Since some members in SA ID are pointers, a physical address to relative address conversion is required before copying this SAD into the variable.

Parameters:

[in]	*SaId*	Pointer of SA ID
[in,out]	*Data*	Pointer of SA Data.

References FIX_REF_BUF_ADDR.

```
VOID FixSpdEntry ( IN EFI_IPSEC_SPD_SELECTOR *   Selector,
                   IN OUT EFI_IPSEC_SPD_DATA *   Data
                 )
```

Fix the value of some members of SPD Selector.

This function is called by IpSecCopyPolicyEntry()which copy the Policy Entry into the Variable. Since some members in SPD Selector are pointers, a physical address to relative address convertion is required before copying this SPD entry into the variable.

Parameters:

[in]	*Selector*	Pointer of SPD Selector.
[in,out]	*Data*	Pointer of SPD Data.

References FIX_REF_BUF_ADDR.

```
EFI_STATUS GetPadEntry ( IN EFI_IPSEC_CONFIG_SELECTOR *   Selector,
                         IN OUT UINTN *                   DataSize,
                         OUT VOID *                       Data
                       )
```

This function lookup the data entry from IPsec PAD. Return the configuration value of the specified PAD Entry.

Parameters:

[in]	*Selector*	Pointer to an entry selector which is an identifier of the PAD entry.
[in,out]	*DataSize*	On output the size of data returned in Data.
[out]	*Data*	The buffer to return the contents of the IPsec configuration data. The type of the data buffer is associated with the DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_NOT_FOUND	The configuration data specified by Selector is not found.
EFI_BUFFER_TOO_SMALL	The DataSize is too small for the result. DataSize has been updated with the size needed to complete the request.

References ComparePadId(), _IPSEC_PAD_ENTRY::Data, _IPSEC_PAD_ENTRY::Id, IPSEC_PAD_ENTRY_FROM_LIST, and mConfigData.

```
EFI_STATUS GetSadEntry ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector,
                         IN OUT UINTN *                  DataSize,
                         OUT VOID *                      Data
                       )
```

This function lookup the data entry from IPsec SAD. Return the configuration value of the specified SAD Entry.

Parameters:

[in]	*Selector*	Pointer to an entry selector which is an identifier of the SAD entry.
[in,out]	*DataSize*	On output, the size of data returned in Data.
[out]	*Data*	The buffer to return the contents of the IPsec configuration data. The type of the data buffer is associated with the DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_NOT_FOUND	The configuration data specified by Selector is not found.
EFI_BUFFER_TOO_SMALL	The DataSize is too small for the result. DataSize has been updated with the size needed to complete the request.

References _IPSEC_SAD_DATA::AlgoInfo, _IPSEC_SAD_DATA::AntiReplayWindowSize, CompareSaId(), _IPSEC_SAD_ENTRY::Data, DuplicateSpdSelector(), _IPSEC_SAD_ENTRY::Id, IPSEC_SAD_ENTRY_FROM_LIST, _IPSEC_SAD_DATA::ManualSet, mConfigData, _IPSEC_SAD_DATA::Mode, _IPSEC_SAD_DATA::PathMTU, _IPSEC_SAD_DATA::SaLifetime, _IPSEC_SAD_DATA::SequenceNumber, SIZE_OF_SPD_SELECTOR, _IPSEC_SAD_DATA::SpdSelector, _IPSEC_SAD_DATA::TunnelDestAddress, and _IPSEC_SAD_DATA::TunnelSourceAddress.

```
EFI_STATUS GetSpdEntry ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector,
                         IN OUT UINTN *                  DataSize,
                         OUT VOID *                      Data
                       )
```

This function lookup the data entry from IPsec SPD. Return the configuration value of the specified SPD Entry.

Parameters:

[in]	*Selector*	Pointer to an entry selector which is an identifier of the SPD entry.
[in,out]	*DataSize*	On output the size of data returned in Data.
[out]	*Data*	The buffer to return the contents of the IPsec configuration data. The type of the data buffer is associated with the DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_INVALID_PARAMETER	Data is NULL and *DataSize is not zero.
EFI_NOT_FOUND	The configuration data specified by Selector is not found.
EFI_BUFFER_TOO_SMALL	The DataSize is too small for the result. DataSize has been updated with the size needed to complete the request.

References _IPSEC_SPD_DATA::Action, CompareSpdSelector(), _IPSEC_SPD_ENTRY::Data, _IPSEC_SAD_ENTRY::Id, IPSEC_SAD_ENTRY_FROM_SPD, IPSEC_SPD_ENTRY_FROM_LIST, IpSecDuplicateProcessPolicy(), IpSecGetSizeOfSpdData(), mConfigData, _IPSEC_SPD_DATA::Name, _IPSEC_SPD_DATA::PackageFlag, _IPSEC_SPD_DATA::ProcessingPolicy, _IPSEC_SPD_DATA::Sas, _IPSEC_SPD_ENTRY::Selector, and _IPSEC_SPD_DATA::TrafficDirection.

EFI_STATUS IpSecConfigInitialize (IN OUT IPSEC_PRIVATE_DATA * *Private*)

Install and Initialize IPsecConfig protocol

Parameters:

`[in,out]` *Private* Pointer to IPSEC_PRIVATE_DATA. After this function finish, the pointer of IPsecConfig Protocol implementation will copy into its IPsecConfig member.

Return values:

EFI_SUCCESS Initialized the IPsecConfig Protocol successfully.
Others Initializing the IPsecConfig Protocol failed.

References IpSecConfigRestore(), and mConfigData.

Referenced by IpSecDriverEntryPoint().

EFI_STATUS IpSecConfigRestore (IN IPSEC_PRIVATE_DATA * *Private*)

Get the all IPSec configuration variables and store those variables to the internal data structure.

This founction is called by IpSecConfigInitialize() that is to intialize the IPsecConfiguration Protocol.

Parameters:

`[in]` *Private* Point to IPSEC_PRIVATE_DATA.

Return values:

EFI_OUT_OF_RESOURCES The required system resource could not be allocated.
EFI_SUCCESS Restore the IPsec Configuration successfully.
others Other errors is found during the variable getting.

Get the all IPSec configuration variables and store those variables to the internal data structure.

This founction is called by IpSecConfigInitialize() which is to intialize the IPsecConfiguration Protocol.

Parameters:

`[in]` *Private* Point to IPSEC_PRIVATE_DATA.

Return values:

EFI_OUT_OF_RESOURCES The required system resource could not be allocated
EFI_SUCCESS Restore the IPsec Configuration successfully.
others Other errors is found while obtaining the variable.

References EfiIpSecConfigSetData(), IPSEC_STATUS_ENABLED, IPSEC_VAR_ITEM_HEADER_CONTENT_BIT, IPSEC_VAR_ITEM_HEADER_LOGO_BIT, IPSECCONFIG_STATUS_NAME, IPSECCONFIG_VARIABLE_NAME, IpSecGetVariable(), mSetBySelf, IPSEC_VAR_ITEM_HEADER::Size, and IPSEC_VAR_ITEM_HEADER::Type.

Referenced by IpSecConfigInitialize().

EFI_STATUS IpSecConfigSave (VOID)

This function is the subfunction of EFIIpSecConfigSetData.

This function call IpSecSetVaraible to set the IPsec Configuration into the firmware.

Return values:

EFI_OUT_OF_RESOURCES	The required system resource could not be allocated.
EFI_SUCCESS	Saved the configration successfully.
Others	Other errors were found while obtaining the variable.

References IPSEC_VARIABLE_BUFFER::Capacity, IPSEC_DEFAULT_VARIABLE_SIZE, IPSECCONFIG_VARIABLE_NAME, IpSecCopyPolicyEntry(), IpSecSetVariable(), IpSecVisitConfigData(), IPSEC_VARIABLE_BUFFER::Ptr, and IPSEC_VARIABLE_BUFFER::Size.

Referenced by EfiIpSecConfigSetData().

```
EFI_STATUS IpSecCopyPolicyEntry ( IN EFI_IPSEC_CONFIG_DATA_TYPE     Type,
                                  IN EFI_IPSEC_CONFIG_SELECTOR *    Selector,
                                  IN VOID *                         Data,
                                  IN UINTN                          SelectorSize,
                                  IN UINTN                          DataSize,
                                  IN OUT IPSEC_VARIABLE_BUFFER *    Buffer
                                )
```

Copy whole data in specified EFI_SIPEC_CONFIG_SELECTOR and the Data to a buffer.

This function is a caller defined function, and it is called by the IpSecVisitConfigData(). The orignal caller is IpSecConfigSave(), which calls the IpsecVisitConfigData() to copy all types of IPsec Config datas into one buffer and store this buffer into firmware in the form of several variables.

Parameters:

`[in]`	*Type*	A specified IPSEC_CONFIG_DATA_TYPE.
`[in]`	*Selector*	Points to a EFI_IPSEC_CONFIG_SELECTOR to be copied to the buffer.
`[in]`	*Data*	Points to data to be copied to the buffer. The Data type is related to the Type.
`[in]`	*SelectorSize*	The size of the Selector.
`[in]`	*DataSize*	The size of the Data.
`[in,out]`	*Buffer*	The buffer to store the Selector and Data.

Return values:

EFI_SUCCESS	Copy the Selector and Data to a buffer successfully.
EFI_OUT_OF_RESOURCES	The required system resource could not be allocated.

References IPSEC_VAR_ITEM_HEADER_LOGO_BIT, IPSEC_VAR_ITEM_HEADER::Size, and IPSEC_VAR_ITEM_HEADER::Type.

Referenced by IpSecConfigSave().

```
VOID IpSecDuplicateProcessPolicy ( IN EFI_IPSEC_PROCESS_POLICY *  Dst,
                                   IN EFI_IPSEC_PROCESS_POLICY *  Src
                                 )
```

Copy Source Process Policy to the Destination Process Policy.

Parameters:

[in] *Dst* Pointer to the Source Process Policy.

[in] *Src* Pointer to the Destination Process Policy.

Referenced by GetSpdEntry(), and SetSpdEntry().

```
UINTN IpSecGetSizeOfEfiSpdData ( IN EFI_IPSEC_SPD_DATA * SpdData )
```

Calculate the a whole size of EFI_IPSEC_SPD_DATA, which includes the buffer size pointed to by the pointer members.

Parameters:

[in] *SpdData* Pointer to a specified EFI_IPSEC_SPD_DATA.

Returns:

the whole size the specified EFI_IPSEC_SPD_DATA.

Referenced by SetSpdEntry().

```
UINTN IpSecGetSizeOfSpdData ( IN IPSEC_SPD_DATA *  SpdData )
```

Calculate the a whole size of IPSEC_SPD_DATA which includes the buffer size pointed to by the pointer members and the buffer size used by the Sa List.

Parameters:

[in] *SpdData* Pointer to the specified IPSEC_SPD_DATA.

Returns:

the whole size of IPSEC_SPD_DATA.

Referenced by GetSpdEntry().

```
EFI_STATUS IpSecGetVariable ( IN CHAR16 *        VariableName,
                              IN EFI_GUID *      VendorGuid,
                              IN UINT32 *        Attributes,
                              IN OUT UINTN *     DataSize,
                              IN VOID *          Data
                            )
```

Get the IPsec Variable.

Get the all variables which start with the string contained in VaraiableName. Since all IPsec related variable store in continual space, those kinds of variable can be searched by the EfiGetNextVariableName. Those variables also are returned in a continual buffer.

Parameters:

[in]	*VariableName*	Pointer to a specified Variable Name.
[in]	*VendorGuid*	Pointer to a specified Vendor Guid.
[in]	*Attributes*	Point to memory location to return the attributes of variable. If the point is NULL, the parameter would be ignored.
[in,out]	*DataSize*	As input, point to the maximum size of return Data-Buffer. As output, point to the actual size of the returned Data-Buffer.
[in]	*Data*	Point to return Data-Buffer.

Return values:

EFI_ABORTED	If the Variable size which contained in the variable structure doesn't match the variable size obtained from the EFIGetVariable.
EFI_BUFFER_TOO_SMALL	The DataSize is too small for the result. DataSize has been updated with the size needed to complete the request.
EFI_SUCCESS	The function completed successfully.
others	Other errors found during the variable getting.

References IP_SEC_VARIABLE_INFO::SingleVariableSize, IP_SEC_VARIABLE_INFO::VariableCount, and IP_SEC_VARIABLE_INFO::VariableSize.

Referenced by IpSecConfigRestore().

```
EFI_STATUS IpSecSetVariable ( IN CHAR16 *     VariableName,
                              IN EFI_GUID *   VendorGuid,
                              IN UINT32       Attributes,
                              IN UINTN        DataSize,
                              IN VOID *       Data
                            )
```

Set the IPsec variables.

Set all IPsec variables which start with the specified variable name. Those variables are set one by one.

Parameters:

[in]	*VariableName*	The name of the vendor's variable. It is a Null-Terminated Unicode String.
[in]	*VendorGuid*	Unify identifier for vendor.
[in]	*Attributes*	Point to memory location to return the attributes of variable. If the point is NULL, the parameter would be ignored.
[in]	*DataSize*	The size in bytes of Data-Buffer.
[in]	*Data*	Points to the content of the variable.

Return values:

EFI_SUCCESS	The firmware successfully stored the variable and its data, as defined by the Attributes.
others	Storing the variables failed.

References IP_SEC_VARIABLE_INFO::SingleVariableSize, IP_SEC_VARIABLE_INFO::VariableCount, and IP_SEC_VARIABLE_INFO::VariableSize.

Referenced by IpSecConfigSave().

```
EFI_STATUS IpSecVisitConfigData ( IN EFI_IPSEC_CONFIG_DATA_TYPE  DataType,
                                  IN IPSEC_COPY_POLICY_ENTRY     Routine,
                                  IN VOID *                      Context
                                )
```

Visit all IPsec Configurations of specified Type and call the caller defined interface.

Parameters:

- [in] *DataType* The specified IPsec Config Data Type.
- [in] *Routine* The function defined by the caller.
- [in] *Context* The data passed to the Routine.

Return values:

- *EFI_OUT_OF_RESOURCES* The required system resource could not be allocated
- *EFI_SUCCESS* This function completed successfully.

References EfiIpSecConfigGetData(), and EfiIpSecConfigGetNextSelector().

Referenced by Ikev2ChildSaAssociateSpdEntry(), and IpSecConfigSave().

```
BOOLEAN IsInAddressInfoList ( IN EFI_IP_ADDRESS_INFO *  AddressInfo,
                              IN EFI_IP_ADDRESS_INFO *  AddressInfoList,
                              IN UINT32                 AddressCount
                            )
```

Check if the specified EFI_IP_ADDRESS_INFO is in EFI_IP_ADDRESS_INFO list.

Parameters:

- [in] *AddressInfo* Pointer of IP_ADDRESS_INFO to be search in AddressInfo list.
- [in] *AddressInfoList* A list that contains IP_ADDRESS_INFOs.
- [in] *AddressCount* Point out how many IP_ADDRESS_INFO in the list.

Return values:

- *TRUE* The specified AddressInfo is in the AddressInfoList.
- *FALSE* The specified AddressInfo is not in the AddressInfoList.

Referenced by CompareSpdSelector(), and IsSubSpdSelector().

```
BOOLEAN IsSubSpdSelector ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector1,
                           IN EFI_IPSEC_CONFIG_SELECTOR *  Selector2
                         )
```

Find if the two SPD Selectors has subordinative.

Compare two SPD Selector by the fields of LocalAddressCount/RemoteAddressCount/NextLayerProtocol/LocalPort/LocalPortRange/RemotePort/RemotePortRange and the Local Addresses and remote Addresses.

Parameters:

- [in] *Selector1* Pointer of first SPD Selector.
- [in] *Selector2* Pointer of second SPD Selector.

Return values:

TRUE	The first SPD Selector is subordinate Selector of second SPD Selector.
FALSE	The first SPD Selector is not subordinate Selector of second SPD Selector.

References IsInAddressInfoList().

Referenced by IpSecProcess(), and SetSadEntry().

BOOLEAN IsZeroPadId (IN EFI_IPSEC_CONFIG_SELECTOR * *Selector*)

Check if the PAD ID is Zero.

Parameters:

`[in]` *Selector* Pointer of the PAD ID.

Return values:

TRUE If the PAD ID is Zero.
FALSE If the PAD ID is not Zero.

BOOLEAN IsZeroSaId (IN EFI_IPSEC_CONFIG_SELECTOR * *Selector*)

Check if the SA ID is Zero by its DestAddress.

Parameters:

`[in]` *Selector* Pointer of the SA ID.

Return values:

TRUE If the SA ID is Zero.
FALSE If the SA ID is not Zero.

BOOLEAN IsZeroSpdSelector (IN EFI_IPSEC_CONFIG_SELECTOR * *Selector*)

Check if the SPD Selector is Zero by its LocalAddressCount and RemoteAddressCount fields.

Parameters:

`[in]` *Selector* Pointer of the SPD Selector.

Return values:

TRUE If the SPD Selector is Zero.
FALSE If the SPD Selector is not Zero.

EFI_STATUS SetPadEntry (IN EFI_IPSEC_CONFIG_SELECTOR * *Selector*,
IN VOID * *Data*,
IN VOID * *Context*
)

Set the peer authorization configuration information for the EFI IPsec driver.

The IPsec configuration data has a unique selector/identifier separately to identify a data entry.

Parameters:

[in] *Selector* Pointer to an entry selector on operated configuration data specified by DataType. A NULL Selector causes the entire specified-type configuration information to be flushed.

[in] *Data* The data buffer to be set. The structure of the data buffer should be EFI_IPSEC_PAD_DATA.

[in] *Context* Pointer to one entry selector that describes the expected position the new data entry will be added. If Context is NULL, the new entry will be appended the end of database.

Return values:

EFI_OUT_OF_RESOURCES The required system resources could not be allocated.

EFI_SUCCESS The specified configuration data was obtained successfully.

References ComparePadId(), _IPSEC_PAD_ENTRY::Data, _IPSEC_PAD_ENTRY::Id, IPSEC_PAD_ENTRY_FROM_LIST, _IPSEC_PAD_ENTRY::List, and mConfigData.

```
EFI_STATUS SetSadEntry ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector,
                         IN VOID *                       Data,
                         IN VOID *                       Context
                       )
```

Set the security association information for the EFI IPsec driver.

The IPsec configuration data has a unique selector/identifier separately to identify a data entry.

Parameters:

[in] *Selector* Pointer to an entry selector on operated configuration data specified by DataType. A NULL Selector causes the entire specified-type configuration information to be flushed.

[in] *Data* The data buffer to be set. The structure of the data buffer should be EFI_IPSEC_SA_DATA.

[in] *Context* Pointer to one entry selector which describes the expected position the new data entry will be added. If Context is NULL,the new entry will be appended the end of database.

Return values:

EFI_OUT_OF_RESOURCED The required system resource could not be allocated.

EFI_SUCCESS The specified configuration data was obtained successfully.

References _IPSEC_SPD_DATA::Action, _IPSEC_SAD_DATA::AlgoInfo, _IPSEC_SAD_DATA::AntiReplayBitmap, _IPSEC_SAD_DATA::AntiReplayWindowSize, _IPSEC_SAD_ENTRY::BySpd, CompareSaId(), _IPSEC_SPD_ENTRY::Data, _IPSEC_SAD_ENTRY::Data, DuplicateSpdSelector(), _IPSEC_SAD_DATA::ESNEnabled, _IPSEC_SAD_ENTRY::Id, IPSEC_SAD_ENTRY_FROM_LIST, IPSEC_SPD_ENTRY_FROM_LIST, IsSubSpdSelector(), _IPSEC_SAD_ENTRY::List, _IPSEC_SAD_DATA::ManualSet, mConfigData, _IPSEC_SAD_DATA::Mode, _IPSEC_SAD_DATA::PathMTU, _IPSEC_SAD_DATA::SaLifetime, _IPSEC_SPD_DATA::Sas, _IPSEC_SPD_ENTRY::Selector, _IPSEC_SAD_DATA::SequenceNumber, SIZE_OF_SPD_SELECTOR, _IPSEC_SAD_DATA::SpdEntry, _IPSEC_SAD_DATA::SpdSelector, _IPSEC_SAD_DATA::TunnelDestAddress, and _IPSEC_SAD_DATA::TunnelSourceAddress.

```
EFI_STATUS SetSpdEntry ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector,
                         IN VOID *                       Data,
```

IN VOID * *Context*
)

Set the security policy information for the EFI IPsec driver.

The IPsec configuration data has a unique selector/identifier separately to identify a data entry.

Parameters:

`[in]` *Selector* Pointer to an entry selector on operated configuration data specified by DataType. A NULL Selector causes the entire specified-type configuration information to be flushed.

`[in]` *Data* The data buffer to be set. The structure of the data buffer should be EFI_IPSEC_SPD_DATA.

`[in]` *Context* Pointer to one entry selector that describes the expected position the new data entry will be added. If Context is NULL, the new entry will be appended the end of database.

Return values:

EFI_INVALID_PARAMETER One or more of the following are TRUE:
- ◊ Selector is not NULL and its LocalAddress is NULL or its RemoteAddress is NULL.
- ◊ Data is not NULL and its Action is Protected and its plolicy is NULL.
- ◊ Data is not NULL, its Action is not protected, and its policy is not NULL.
- ◊ The Action of Data is Protected, its policy mode is Tunnel, and its tunnel option is NULL.
- ◊ The Action of Data is protected and its policy mode is not Tunnel and it tunnel option is not NULL.

EFI_OUT_OF_RESOURCED The required system resource could not be allocated.

EFI_SUCCESS The specified configuration data was obtained successfully.

References _IPSEC_SPD_DATA::Action, _IPSEC_SAD_ENTRY::BySpd, CompareSaId(), CompareSpdSelector(), _IPSEC_SAD_ENTRY::Data, _IPSEC_SPD_ENTRY::Data, DuplicateSpdSelector(), _IPSEC_SAD_ENTRY::Id, IPSEC_SAD_ENTRY_FROM_LIST, IPSEC_SAD_ENTRY_FROM_SPD, IPSEC_SPD_ENTRY_FROM_LIST, IpSecDuplicateProcessPolicy(), IpSecGetSizeOfEfiSpdData(), _IPSEC_SPD_ENTRY::List, mConfigData, _IPSEC_SPD_DATA::Name, _IPSEC_SPD_DATA::PackageFlag, _IPSEC_SPD_DATA::ProcessingPolicy, _IPSEC_SPD_DATA::Sas, _IPSEC_SPD_ENTRY::Selector, SIZE_OF_SPD_SELECTOR, _IPSEC_SAD_DATA::SpdEntry, and _IPSEC_SPD_DATA::TrafficDirection.

VOID UnfixPadEntry (IN EFI_IPSEC_PAD_ID * *PadId*,
IN OUT EFI_IPSEC_PAD_DATA * *Data*
)

Recover the value of some members of PAD ID.

This function is corresponding to FixPadEntry(). It recovers the value of members of PAD ID that are fixed by FixPadEntry().

Parameters:

`[in]` *PadId* Pointer of PAD ID.

`[in,out]` *Data* Pointer of PAD Data.

References UNFIX_REF_BUF_ADDR.

VOID UnfixSadEntry (IN OUT EFI_IPSEC_SA_ID * *SaId*,
IN OUT EFI_IPSEC_SA_DATA2 * *Data*
)

Recover the value of some members of SA ID.

This function is corresponding to FixSadEntry(). It recovers the value of members of SAD ID that are fixed by FixSadEntry().

Parameters:

[in,out]	*SaId*	Pointer of SAD ID.
[in,out]	*Data*	Pointer of SAD Data.

References UNFIX_REF_BUF_ADDR.

VOID UnfixSpdEntry (IN OUT EFI_IPSEC_SPD_SELECTOR * *Selector*,
IN OUT EFI_IPSEC_SPD_DATA * *Data*
)

Recover the value of some members of SPD Selector.

This function is corresponding to FixSpdEntry(). It recovers the value of members of SPD Selector that are fixed by FixSpdEntry().

Parameters:

[in,out]	*Selector*	Pointer of SPD Selector.
[in,out]	*Data*	Pointer of SPD Data.

References UNFIX_REF_BUF_ADDR.

Variable Documentation

IPSEC_COMPARE_SELECTOR mCompareSelector[]

Initial value:

```
{
  (IPSEC_COMPARE_SELECTOR) CompareSpdSelector,
  (IPSEC_COMPARE_SELECTOR) CompareSaId,
  (IPSEC_COMPARE_SELECTOR) ComparePadId
}
```

Referenced by EfiIpSecConfigGetNextSelector().

LIST_ENTRY mConfigData[IPsecConfigDataTypeMaximum]

Referenced by EfiIpSecConfigGetNextSelector(), GetPadEntry(), GetSadEntry(), GetSpdEntry(), IkeSearchSpdEntry(), IpSecConfigInitialize(), IpSecLookupPadEntry(), IpSecLookupSadBySpi(), IpSecProcess(), SetPadEntry(), SetSadEntry(), and SetSpdEntry().

IPSEC_DUPLICATE_SELECTOR mDuplicateSelector[]

Initial value:

```
{
  (IPSEC_DUPLICATE_SELECTOR) DuplicateSpdSelector,
  (IPSEC_DUPLICATE_SELECTOR) DuplicateSaId,
  (IPSEC_DUPLICATE_SELECTOR) DuplicatePadId
}
```

IPSEC_FIX_POLICY_ENTRY mFixPolicyEntry[]

Initial value:

```
{
  (IPSEC_FIX_POLICY_ENTRY) FixSpdEntry,
  (IPSEC_FIX_POLICY_ENTRY) FixSadEntry,
  (IPSEC_FIX_POLICY_ENTRY) FixPadEntry
}
```

IPSEC_GET_POLICY_ENTRY mGetPolicyEntry[]

Initial value:

```
{
  (IPSEC_GET_POLICY_ENTRY) GetSpdEntry,
  (IPSEC_GET_POLICY_ENTRY) GetSadEntry,
  (IPSEC_GET_POLICY_ENTRY) GetPadEntry
}
```

EFI_IPSEC_CONFIG_PROTOCOL mIpSecConfigInstance

Initial value:

```
{
  EfiIpSecConfigSetData,
  EfiIpSecConfigGetData,
  EfiIpSecConfigGetNextSelector,
  EfiIpSecConfigRegisterNotify,
  EfiIpSecConfigUnregisterNotify
}
```

IPSEC_IS_ZERO_SELECTOR mIsZeroSelector[]

Initial value:

```
{
  (IPSEC_IS_ZERO_SELECTOR) IsZeroSpdSelector,
  (IPSEC_IS_ZERO_SELECTOR) IsZeroSaId,
  (IPSEC_IS_ZERO_SELECTOR) IsZeroPadId
}
```

BOOLEAN mSetBySelf = FALSE

Referenced by EfiIpSecConfigSetData(), and IpSecConfigRestore().

IPSEC_SET_POLICY_ENTRY mSetPolicyEntry[]

Initial value:

```
{
  (IPSEC_SET_POLICY_ENTRY) SetSpdEntry,
  (IPSEC_SET_POLICY_ENTRY) SetSadEntry,
  (IPSEC_SET_POLICY_ENTRY) SetPadEntry
}
```

IPSEC_FIX_POLICY_ENTRY mUnfixPolicyEntry[]

Initial value:

```
{
  (IPSEC_FIX_POLICY_ENTRY) UnfixSpdEntry,
  (IPSEC_FIX_POLICY_ENTRY) UnfixSadEntry,
  (IPSEC_FIX_POLICY_ENTRY) UnfixPadEntry
}
```

Generated on Mon Sep 28 08:49:55 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/IpSecConfigImpl.h File Reference

Data Structures

struct IP_SEC_VARIABLE_INFO
struct IPSEC_COMMON_POLICY_ENTRY
struct IPSEC_VARIABLE_BUFFER
struct IPSEC_VAR_ITEM_HEADER

Defines

"memItemRight" valign="bottom">EFI_IPSEC_ANY_PROTOCOL 0xFFFF
#define EFI_IPSEC_ANY_PORT 0
#define IPSEC_VAR_ITEM_HEADER_LOG
#define IPSEC_VAR_ITEM_HEADER_CON
#define IPSECCONFIG_VARIABLE_NAME
#define IPSECCONFIG_STATUS_NAME L
#define SIZE_OF_SPD_SELECTOR(x)
#define FIX_REF_BUF_ADDR(addr, base) a
((UINTN) (addr) - (UINTN) (base))
#define UNFIX_REF_BUF_ADDR(addr, base
*) ((UINTN) (addr) + (UINTN) (base))

Typedefs

typedef EFI_STATUS(* IPSEC_DUPLICATE_SELECTOR)(I
EFI_IPSEC_CONFIG_SELECTOR *D
EFI_IPSEC_CONFIG_SELECTOR *S
UINTN *Size)
typedef BOOLEAN(* IPSEC_COMPARE_SELECTOR)(IN
EFI_IPSEC_CONFIG_SELECTOR *S
EFI_IPSEC_CONFIG_SELECTOR *S
typedef BOOLEAN(* IPSEC_IS_ZERO_SELECTOR)(IN
EFI_IPSEC_CONFIG_SELECTOR *S
typedef VOID(* IPSEC_FIX_POLICY_ENTRY)(IN
EFI_IPSEC_CONFIG_SELECTOR *S
*Data)
typedef EFI_STATUS(* IPSEC_COPY_POLICY_ENTRY)(IN
EFI_IPSEC_CONFIG_DATA_TYPE
EFI_IPSEC_CONFIG_SELECTOR *S
*Data, IN UINTN SelectorSize, IN UI
OUT VOID *Context)
typedef EFI_STATUS(* IPSEC_SET_POLICY_ENTRY)(IN
EFI_IPSEC_CONFIG_SELECTOR *S
*Data, IN VOID *Context)
typedef EFI_STATUS(* IPSEC_GET_POLICY_ENTRY)(IN
EFI_IPSEC_CONFIG_SELECTOR *S
UINTN *DataSize, IN VOID *Data)

Functions

BOOLEAN CompareSpdSelector (IN EFI_IPSEC_CONFIG_SELECTOR *Selector1 EFI_IPSEC_CONFIG_SELECTOR *Selector2

EFI_STATUS IpSecVisitConfigData (IN EFI_IPSEC_CONFIG_DATA_TYPE DataTyp IPSEC_COPY_POLICY_ENTRY Routine, IN *Context)

EFI_STATUS IpSecConfigSave (VOID)

EFI_STATUS IpSecConfigInitialize (IN OUT IPSEC_PRIVATE_DATA *Private)

UINTN IpSecGetSizeOfEfiSpdData (IN EFI_IPSEC_SPD_DATA *SpdData)

UINTN IpSecGetSizeOfSpdData (IN IPSEC_SPD_DA *SpdData)

VOID IpSecDuplicateProcessPolicy (IN EFI_IPSEC_PROCESS_POLICY *Dst, IN EFI_IPSEC_PROCESS_POLICY *Src)

BOOLEAN IsSubSpdSelector (IN EFI_IPSEC_CONFIG_SELECTOR *Selector1 EFI_IPSEC_CONFIG_SELECTOR *Selector2

BOOLEAN CompareSaId (IN EFI_IPSEC_CONFIG_SEL *Selector1, IN EFI_IPSEC_CONFIG_SELECT *Selector2)

BOOLEAN ComparePadId (IN EFI_IPSEC_CONFIG_SEL *Selector1, IN EFI_IPSEC_CONFIG_SELECT *Selector2)

BOOLEAN IsZeroSpdSelector (IN EFI_IPSEC_CONFIG_SELECTOR *Selector)

BOOLEAN IsZeroSaId (IN EFI_IPSEC_CONFIG_SELEC *Selector)

BOOLEAN IsZeroPadId (IN EFI_IPSEC_CONFIG_SELE *Selector)

EFI_STATUS DuplicateSpdSelector (IN OUT EFI_IPSEC_CONFIG_SELECTOR *DstSel, I EFI_IPSEC_CONFIG_SELECTOR *SrcSel, I UINTN *Size)

EFI_STATUS DuplicateSaId (IN OUT EFI_IPSEC_CONFIG_SELECTOR *DstSel, I EFI_IPSEC_CONFIG_SELECTOR *SrcSel, I UINTN *Size)

EFI_STATUS DuplicatePadId (IN OUT EFI_IPSEC_CONFIG_SELECTOR *DstSel, I EFI_IPSEC_CONFIG_SELECTOR *SrcSel, I UINTN *Size)

VOID FixSpdEntry (IN EFI_IPSEC_SPD_SELECTO *Selector, IN OUT EFI_IPSEC_SPD_DATA *

VOID FixSadEntry (IN EFI_IPSEC_SA_ID *SaId, IN EFI_IPSEC_SA_DATA2 *Data)

VOID FixPadEntry (IN EFI_IPSEC_PAD_ID *PadId, EFI_IPSEC_PAD_DATA *Data)

VOID UnfixSpdEntry (IN OUT EFI_IPSEC_SPD_SE *Selector, IN OUT EFI_IPSEC_SPD_DATA *

VOID

UnfixSadEntry (IN OUT EFI_IPSEC_S
OUT EFI_IPSEC_SA_DATA2 *Data)
VOID UnfixPadEntry (IN EFI_IPSEC_PAD_
OUT EFI_IPSEC_PAD_DATA *Data)
EFI_STATUS SetSpdEntry (IN EFI_IPSEC_CONFIG
*Selector, IN VOID *Data, IN VOID *
EFI_STATUS SetSadEntry (IN EFI_IPSEC_CONFIG
*Selector, IN VOID *Data, IN VOID *
EFI_STATUS SetPadEntry (IN EFI_IPSEC_CONFIG
*Selector, IN VOID *Data, IN VOID *
EFI_STATUS GetSpdEntry (IN EFI_IPSEC_CONFIG
*Selector, IN OUT UINTN *DataSize,
*Data)
EFI_STATUS GetSadEntry (IN EFI_IPSEC_CONFIG
*Selector, IN OUT UINTN *DataSize,
*Data)
EFI_STATUS GetPadEntry (IN EFI_IPSEC_CONFIG
*Selector, IN OUT UINTN *DataSize,
*Data)
EFI_STATUS EFIAPI EfiIpSecConfigGetData (IN
EFI_IPSEC_CONFIG_PROTOCOL *
EFI_IPSEC_CONFIG_DATA_TYPE
EFI_IPSEC_CONFIG_SELECTOR *S
UINTN *DataSize, OUT VOID *Data)
EFI_STATUS EFIAPI EfiIpSecConfigSetData (IN
EFI_IPSEC_CONFIG_PROTOCOL *
EFI_IPSEC_CONFIG_DATA_TYPE
EFI_IPSEC_CONFIG_SELECTOR *S
*Data, IN EFI_IPSEC_CONFIG_SELE
*InsertBefore)
EFI_STATUS EFIAPI EfiIpSecConfigGetNextSelector (IN
EFI_IPSEC_CONFIG_PROTOCOL *
EFI_IPSEC_CONFIG_DATA_TYPE
OUT UINTN *SelectorSize, IN OUT
EFI_IPSEC_CONFIG_SELECTOR *S
EFI_STATUS EFIAPI EfiIpSecConfigRegisterNotify (IN
EFI_IPSEC_CONFIG_PROTOCOL *
EFI_IPSEC_CONFIG_DATA_TYPE
EFI_EVENT Event)
EFI_STATUS EFIAPI EfiIpSecConfigUnregisterNotify (IN
EFI_IPSEC_CONFIG_PROTOCOL *
EFI_IPSEC_CONFIG_DATA_TYPE
EFI_EVENT Event)

Variables

LIST_ENTRY mConfigData [IPsecConfigDataTypeM

Detailed Description

Definitions related to IPSEC_CONFIG_PROTOCOL implementations.

Copyright (c) 2009 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

#define EFI_IPSEC_ANY_PROTOCOL 0xFFFF

#define FIX_REF_BUF_ADDR (addr, base) addr = (VOID *) ((UINTN) (addr) - (UINTN) (base))

Referenced by FixPadEntry(), FixSadEntry(), and FixSpdEntry().

#define IPSEC_VAR_ITEM_HEADER_CONTENT_BIT 0x7F

Referenced by IpSecConfigRestore().

#define IPSEC_VAR_ITEM_HEADER_LOGO_BIT 0x80

Referenced by IpSecConfigRestore(), and IpSecCopyPolicyEntry().

#define IPSECCONFIG_STATUS_NAME L"IpSecStatus"

#define IPSECCONFIG_VARIABLE_NAME L"IpSecConfig"

Referenced by IpSecConfigRestore(), and IpSecConfigSave().

#define SIZE_OF_SPD_SELECTOR (x)

Value:

```
(UINTN) (sizeof (EFI_IPSEC_SPD_SELECTOR) \
       + sizeof (EFI_IP_ADDRESS_INFO) * ((x)->LocalAddressCount + (x)->RemoteAddressCount))
```

Referenced by DuplicateSpdSelector(), GetSadEntry(), SetSadEntry(), and SetSpdEntry().

#define UNFIX_REF_BUF_ADDR (addr, base) addr = (VOID *) ((UINTN) (addr) + (UINTN) (base))

Referenced by UnfixPadEntry(), UnfixSadEntry(), and UnfixSpdEntry().

Typedef Documentation

typedef BOOLEAN(* IPSEC_COMPARE_SELECTOR)(IN EFI_IPSEC_CONFIG_SELECTOR *Selector1, IN EFI_IPSEC_CONFIG_SELECTOR *Selector2)

It is prototype of compare two Selectors. The Selector would be SPD Selector, or SAD Selector, or PAD selector.

Parameters:

[in]	*Selector1*	Pointer of the first Selector.
[in]	*Selector2*	Pointer of the second Selector.

Return values:

TRUE	These two Selectors have the same value in certain fields.
FALSE	Not all fields have the same value in these two Selectors.

typedef EFI_STATUS(* IPSEC_COPY_POLICY_ENTRY)(IN EFI_IPSEC_CONFIG_DATA_TYPE Type, IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data, IN UINTN SelectorSize, IN UINTN DataSize, IN OUT VOID *Context)

It is prototype function to define a routine function by the caller of IpSecVisitConfigData().

Parameters:

[in]	*Type*	A specified IPSEC_CONFIG_DATA_TYPE.
[in]	*Selector*	Points to EFI_IPSEC_CONFIG_SELECTOR to be copied to the buffer.
[in]	*Data*	Points to data to be copied to the buffer. The Data type is related to the Type.
[in]	*SelectorSize*	The size of the Selector.
[in]	*DataSize*	The size of the Data.
[in,out]	*Buffer*	The buffer to store the Selector and Data.

Return values:

EFI_SUCCESS	Copied the Selector and Data to a buffer successfully.
EFI_OUT_OF_RESOURCES	The required system resource could not be allocated.

typedef EFI_STATUS(* IPSEC_DUPLICATE_SELECTOR)(IN OUT EFI_IPSEC_CONFIG_SELECTOR *DstSel, IN EFI_IPSEC_CONFIG_SELECTOR *SrcSel, IN OUT UINTN *Size)

The prototype of Copy Source Selector to the Destination Selector.

Parameters:

[in,out]	*DstSel*	Pointer of Destination Selector. It would be SPD Selector, or SAD Selector or PAD Selector.
[in]	*SrcSel*	Pointer of Source Selector. It would be SPD Selector, or SAD Selector or PAD Selector.
[in,out]	*Size*	The size of the Destination Selector. If it is not NULL and its value is less than the size of Source Selector, the value of Source Selector's size will be passed to the caller by this parameter.

Return values:

EFI_INVALID_PARAMETER	If the Destination or Source Selector is NULL.

EFI_BUFFER_TOO_SMALL	If the input Size is less than size of Source Selector.
EFI_SUCCESS	Copy Source Selector to the Destination Selector successfully.

typedef VOID(* IPSEC_FIX_POLICY_ENTRY)(IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data)

The prototype of a function to fix the value of particular members of the Selector.

Parameters:

[in]	*Selector*	Pointer of Selector.
[in]	*Data*	Pointer of Data.

typedef EFI_STATUS(* IPSEC_GET_POLICY_ENTRY)(IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN OUT UINTN *DataSize, IN VOID *Data)

A prototype function definition to lookup the data entry from IPsec. Return the configuration value of the specified Entry.

Parameters:

[in]	*Selector*	Pointer to an entry selector that is an identifier of the entry.
[in,out]	*DataSize*	On output, the size of data returned in Data.
[out]	*Data*	The buffer to return the contents of the IPsec configuration data. The type of the data buffer is associated with the DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_INVALID_PARAMETER	Data is NULL and *DataSize is not zero.
EFI_NOT_FOUND	The configuration data specified by Selector is not found.
EFI_BUFFER_TOO_SMALL	The DataSize is too small for the result. DataSize has been updated with the size needed to complete the request.

typedef BOOLEAN(* IPSEC_IS_ZERO_SELECTOR)(IN EFI_IPSEC_CONFIG_SELECTOR *Selector)

The prototype of a function to check if the Selector is Zero by its certain fields.

Parameters:

[in]	*Selector*	Pointer of the Selector.

Return values:

TRUE	If the Selector is Zero.
FALSE	If the Selector is not Zero.

typedef EFI_STATUS(* IPSEC_SET_POLICY_ENTRY)(IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data, IN VOID *Context)

Set the security policy information for the EFI IPsec driver.

The IPsec configuration data has a unique selector/identifier separately to identify a data entry.

Parameters:

[in]	*Selector*	Pointer to an entry selector on operated configuration data specified by DataType. A NULL Selector causes the entire specified-type configuration information to be flushed.

[in] *Data* The data buffer to be set.

[in] *Context* Pointer to one entry selector that describes the expected position the new data entry will be added. If Context is NULL, the new entry will be appended to the end of the database.

Return values:

EFI_INVALID_PARAMETER Certain Parameters are not correct. The Parameter requiring a check depends on the Selector type.

EFI_OUT_OF_RESOURCED The required system resource could not be allocated.

EFI_SUCCESS The specified configuration data was obtained successfully.

Function Documentation

```
BOOLEAN ComparePadId ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector1,
                       IN EFI_IPSEC_CONFIG_SELECTOR *  Selector2
                     )
```

Compare two PAD IDs.

Parameters:

[in] *Selector1* Pointer of the first PAD ID.

[in] *Selector2* Pointer of the second PAD ID.

Return values:

TRUE This two Selectors have the same PAD ID.

FALSE This two Selecotrs don't have the same PAD ID.

Compare two PAD IDs.

Parameters:

[in] *Selector1* Pointer of first PAD ID.

[in] *Selector2* Pointer of second PAD ID.

Return values:

TRUE This two Selectors have the same PAD ID.

FALSE This two Selecotrs don't have the same PAD ID.

Referenced by GetPadEntry(), and SetPadEntry().

```
BOOLEAN CompareSaId ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector1,
                      IN EFI_IPSEC_CONFIG_SELECTOR *  Selector2
                    )
```

Compare two SA IDs.

Parameters:

[in] *Selector1* Pointer of the first SA ID.

[in] *Selector2* Pointer of the second SA ID.

Return values:

TRUE This two Selectors have the same SA ID.

FALSE This two Selecotrs don't have the same SA ID.

Compare two SA IDs.

Parameters:

[in] *Selector1* Pointer of first SA ID.
[in] *Selector2* Pointer of second SA ID.

Return values:

TRUE This two Selectors have the same SA ID.
FALSE This two Selecotrs don't have the same SA ID.

Referenced by GetSadEntry(), SetSadEntry(), and SetSpdEntry().

```
BOOLEAN CompareSpdSelector ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector1,
                             IN EFI_IPSEC_CONFIG_SELECTOR *  Selector2
                           )
```

Compare two SPD Selectors.

Compare two SPD Selector by the fields of LocalAddressCount/RemoteAddressCount/NextLayerProtocol/LocalPort/LocalPortRange/RemotePort/RemotePortRange and the Local Addresses and remote Addresses.

Parameters:

[in] *Selector1* Pointer of the first SPD Selector.
[in] *Selector2* Pointer of the second SPD Selector.

Return values:

TRUE These two Selectors have the same value in above fields.
FALSE Not all of the above fields have the same value in these two Selectors.

Compare two SPD Selectors.

Compare two SPD Selector by the fields of LocalAddressCount/RemoteAddressCount/NextLayerProtocol/LocalPort/LocalPortRange/RemotePort/RemotePortRange and the Local Addresses and remote Addresses.

Parameters:

[in] *Selector1* Pointer of first SPD Selector.
[in] *Selector2* Pointer of second SPD Selector.

Return values:

TRUE This two Selector have the same value in above fields.
FALSE Not all above fields have the same value in these two Selectors.

References IsInAddressInfoList().

Referenced by GetSpdEntry(), IkeSearchSpdEntry(), and SetSpdEntry().

```
EFI_STATUS DuplicatePadId ( IN OUT EFI_IPSEC_CONFIG_SELECTOR *  DstSel,
                            IN EFI_IPSEC_CONFIG_SELECTOR *      SrcSel,
                            IN OUT UINTN *                      Size
```

)

Copy Source PAD ID to the Destination PAD ID.

Parameters:

[in,out]	*DstSel*	Pointer of Destination PAD ID.
[in]	*SrcSel*	Pointer of Source PAD ID.
[in,out]	*Size*	The size of the Destination PAD ID. If it not NULL, and its value less than the size of Source PAD ID, the value of Source PAD ID's size will be passed to the caller by this parameter.

Return values:

EFI_INVALID_PARAMETER	If the Destination or Source PAD ID is NULL.
EFI_BUFFER_TOO_SMALL	If the input Size less than size of source PAD ID.
EFI_SUCCESS	Copied Source PAD ID to the Destination PAD ID successfully.

Copy Source PAD ID to the Destination PAD ID.

Parameters:

[in,out]	*DstSel*	Pointer of Destination PAD ID.
[in]	*SrcSel*	Pointer of Source PAD ID.
[in,out]	*Size*	The size of the Destination PAD ID. If it not NULL and its value less than the size of Source PAD ID, the value of Source PAD ID's size will be passed to caller by this parameter.

Return values:

EFI_INVALID_PARAMETER	If the Destination or Source PAD ID is NULL.
EFI_BUFFER_TOO_SMALL	If the input Size less than size of source PAD ID .
EFI_SUCCESS	Copy Source PAD ID to the Destination PAD ID successfully.

```
EFI_STATUS DuplicateSaId ( IN OUT EFI_IPSEC_CONFIG_SELECTOR *   DstSel,
                           IN EFI_IPSEC_CONFIG_SELECTOR *       SrcSel,
                           IN OUT UINTN *                       Size
                         )
```

Copy Source SA ID to the Destination SA ID.

Parameters:

[in,out]	*DstSel*	Pointer of the Destination SA ID.
[in]	*SrcSel*	Pointer of the Source SA ID.
[in,out]	*Size*	The size of the Destination SA ID. If it not NULL, and its value is less than the size of Source SA ID, the value of Source SA ID's size will be passed to the caller by this parameter.

Return values:

EFI_INVALID_PARAMETER	If the Destination or Source SA ID is NULL.
EFI_BUFFER_TOO_SMALL	If the input Size less than size of source SA ID.
EFI_SUCCESS	Copied Source SA ID to the Destination SA ID successfully.

Copy Source SA ID to the Destination SA ID.

Parameters:

[in,out]	*DstSel*	Pointer of Destination SA ID.

[in]	*SrcSel*	Pointer of Source SA ID.
[in,out]	*Size*	The size of the Destination SA ID. If it not NULL and its value less than the size of Source SA ID, the value of Source SA ID's size will be passed to caller by this parameter.

Return values:

EFI_INVALID_PARAMETER	If the Destination or Source SA ID is NULL.
EFI_BUFFER_TOO_SMALL	If the input Size less than size of source SA ID.
EFI_SUCCESS	Copy Source SA ID to the Destination SA ID successfully.

```
EFI_STATUS DuplicateSpdSelector ( IN OUT EFI_IPSEC_CONFIG_SELECTOR *   DstSel,
                                  IN EFI_IPSEC_CONFIG_SELECTOR *       SrcSel,
                                  IN OUT UINTN *                       Size
                                )
```

Copy Source SPD Selector to the Destination SPD Selector.

Parameters:

[in,out]	*DstSel*	Pointer of Destination SPD Selector.
[in]	*SrcSel*	Pointer of Source SPD Selector.
[in,out]	*Size*	The size of the Destination SPD Selector. If it is not NULL and its value is less than the size of Source SPD Selector, the value of Source SPD Selector's size will be passed to the caller by this parameter.

Return values:

EFI_INVALID_PARAMETER	If the Destination or Source SPD Selector is NULL.
EFI_BUFFER_TOO_SMALL	If the input Size is less than size of Source SPD Selector.
EFI_SUCCESS	Copy Source SPD Selector to the Destination SPD Selector successfully.

Copy Source SPD Selector to the Destination SPD Selector.

Parameters:

[in,out]	*DstSel*	Pointer of Destination SPD Selector.
[in]	*SrcSel*	Pointer of Source SPD Selector.
[in,out]	*Size*	The size of the Destination SPD Selector. If it not NULL and its value less than the size of Source SPD Selector, the value of Source SPD Selector's size will be passed to caller by this parameter.

Return values:

EFI_INVALID_PARAMETER	If the Destination or Source SPD Selector is NULL
EFI_BUFFER_TOO_SMALL	If the input Size is less than size of the Source SPD Selector.
EFI_SUCCESS	Copy Source SPD Selector to the Destination SPD Selector successfully.

References SIZE_OF_SPD_SELECTOR.

Referenced by GetSadEntry(), SetSadEntry(), and SetSpdEntry().

```
EFI_STATUS EFIAPI EfiIpSecConfigGetData ( IN EFI_IPSEC_CONFIG_PROTOCOL *   This,
                                          IN EFI_IPSEC_CONFIG_DATA_TYPE    DataType,
```

```
                                    IN EFI_IPSEC_CONFIG_SELECTOR *   Selector,
                                    IN OUT UINTN *                   DataSize,
                                    OUT VOID *                       Data
                                  )
```

Return the configuration value for the EFI IPsec driver.

This function lookup the data entry from IPsec database or IKEv2 configuration information. The expected data type and unique identification are described in DataType and Selector parameters.

Parameters:

`[in]`	*This*	Pointer to the EFI_IPSEC_CONFIG_PROTOCOL instance.
`[in]`	*DataType*	The type of data to retrieve.
`[in]`	*Selector*	Pointer to an entry selector that is an identifier of the IPsec configuration data entry.
`[in,out]`	*DataSize*	On output the size of data returned in Data.
`[out]`	*Data*	The buffer to return the contents of the IPsec configuration data. The type of the data buffer is associated with the DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_INVALID_PARAMETER	One or more of the followings are TRUE: ◊ This is NULL. ◊ Selector is NULL. ◊ DataSize is NULL. ◊ Data is NULL and *DataSize is not zero
EFI_NOT_FOUND	The configuration data specified by Selector is not found.
EFI_UNSUPPORTED	The specified DataType is not supported.
EFI_BUFFER_TOO_SMALL	The DataSize is too small for the result. DataSize has been updated with the size needed to complete the request.

Return the configuration value for the EFI IPsec driver.

This function lookup the data entry from IPsec database or IKEv2 configuration information. The expected data type and unique identification are described in DataType and Selector parameters.

Parameters:

`[in]`	*This*	Pointer to the EFI_IPSEC_CONFIG_PROTOCOL instance.
`[in]`	*DataType*	The type of data to retrieve.
`[in]`	*Selector*	Pointer to an entry selector that is an identifier of the IPsec configuration data entry.
`[in,out]`	*DataSize*	On output the size of data returned in Data.
`[out]`	*Data*	The buffer to return the contents of the IPsec configuration data. The type of the data buffer associated with the DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_INVALID_PARAMETER	One or more of the followings are TRUE: ◊ This is NULL. ◊ Selector is NULL. ◊ DataSize is NULL. ◊ Data is NULL and *DataSize is not zero
EFI_NOT_FOUND	The configuration data specified by Selector is not found.

EFI_UNSUPPORTED	The specified DataType is not supported.
EFI_BUFFER_TOO_SMALL	The DataSize is too small for the result. DataSize has been updated with the size needed to complete the request.

Referenced by IpSecVisitConfigData().

```
EFI_STATUS EFIAPI
EfiIpSecConfigGetNextSelector
                                ( IN EFI_IPSEC_CONFIG_PROTOCOL *     This,
                                  IN EFI_IPSEC_CONFIG_DATA_TYPE      DataType,
                                  IN OUT UINTN *                     SelectorSize,
                                  IN OUT
                                  EFI_IPSEC_CONFIG_SELECTOR *        Selector
                                )
```

Enumerates the current selector for IPsec configuration data entry.

This function is called multiple times to retrieve the entry Selector in IPsec configuration database. On each call to GetNextSelector(), the next entry Selector are retrieved into the output interface.

If the entire IPsec configuration database has been iterated, the error EFI_NOT_FOUND is returned. If the Selector buffer is too small for the next Selector copy, an EFI_BUFFER_TOO_SMALL error is returned, and SelectorSize is updated to reflect the size of buffer needed.

On the initial call to GetNextSelector() to start the IPsec configuration database search, a pointer to the buffer with all zero value is passed in Selector. Calls to SetData() between calls to GetNextSelector may produce unpredictable results.

Parameters:

`[in]`	*This*	Pointer to the EFI_IPSEC_CONFIG_PROTOCOL instance.
`[in]`	*DataType*	The type of IPsec configuration data to retrieve.
`[in,out]`	*SelectorSize*	The size of the Selector buffer.
`[in,out]`	*Selector*	On input, supplies the pointer to last Selector that was returned by GetNextSelector(). On output, returns one copy of the current entry Selector of a given DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_INVALID_PARAMETER	One or more of the followings are TRUE: ◊ This is NULL. ◊ SelectorSize is NULL. ◊ Selector is NULL.
EFI_NOT_FOUND	The next configuration data entry was not found.
EFI_UNSUPPORTED	The specified DataType is not supported.
EFI_BUFFER_TOO_SMALL	The SelectorSize is too small for the result. This parameter has been updated with the size needed to complete the search request.

Enumerates the current selector for IPsec configuration data entry.

This function is called multiple times to retrieve the entry Selector in IPsec configuration database. On each call to GetNextSelector(), the next entry Selector are retrieved into the output interface.

If the entire IPsec configuration database has been iterated, the error EFI_NOT_FOUND is returned. If the Selector buffer is too small for the next Selector copy, an EFI_BUFFER_TOO_SMALL error is returned, and

SelectorSize is updated to reflect the size of buffer needed.

On the initial call to GetNextSelector() to start the IPsec configuration database search, a pointer to the buffer with all zero value is passed in Selector. Calls to SetData() between calls to GetNextSelector may produce unpredictable results.

Parameters:

`[in]`	*This*	Pointer to the EFI_IPSEC_CONFIG_PROTOCOL instance.
`[in]`	*DataType*	The type of IPsec configuration data to retrieve.
`[in,out]`	*SelectorSize*	The size of the Selector buffer.
`[in,out]`	*Selector*	On input, supplies the pointer to last Selector that was returned by GetNextSelector(). On output, returns one copy of the current entry Selector of a given DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_INVALID_PARAMETER	One or more of the followings are TRUE: ◊ This is NULL. ◊ SelectorSize is NULL. ◊ Selector is NULL.
EFI_NOT_FOUND	The next configuration data entry was not found.
EFI_UNSUPPORTED	The specified DataType is not supported.
EFI_BUFFER_TOO_SMALL	The SelectorSize is too small for the result. This parameter has been updated with the size needed to complete the search request.

References mCompareSelector, mConfigData, and IPSEC_COMMON_POLICY_ENTRY::Selector.

Referenced by Ikev2ChildSaSilentDelete(), and IpSecVisitConfigData().

```
EFI_STATUS EFIAPI
EfiIpSecConfigRegisterNotify
                                    ( IN EFI_IPSEC_CONFIG_PROTOCOL *  This,
                                      IN EFI_IPSEC_CONFIG_DATA_TYPE   DataType,
                                      IN EFI_EVENT                    Event
                                    )
```

Register an event that is to be signaled whenever a configuration process on the specified IPsec configuration information is done.

The register function is not surpport now and always returns EFI_UNSUPPORTED.

Parameters:

`[in]`	*This*	Pointer to the EFI_IPSEC_CONFIG_PROTOCOL instance.
`[in]`	*DataType*	The type of data to be registered the event for.
`[in]`	*Event*	The event to be registered.

Return values:

EFI_SUCCESS	The event is registered successfully.
EFI_INVALID_PARAMETER	This is NULL, or Event is NULL.
EFI_ACCESS_DENIED	The Event is already registered for the DataType.
EFI_UNSUPPORTED	The notify registration unsupported, or the specified DataType is not supported.

Register an event that is to be signaled whenever a configuration process on the specified IPsec configuration information is done.

The register function is not surpport now and always returns EFI_UNSUPPORTED.

Parameters:

[in]	*This*	Pointer to the EFI_IPSEC_CONFIG_PROTOCOL instance.
[in]	*DataType*	The type of data to be registered the event for.
[in]	*Event*	The event to be registered.

Return values:

EFI_SUCCESS	The event is registered successfully.
EFI_INVALID_PARAMETER	This is NULL or Event is NULL.
EFI_ACCESS_DENIED	The Event is already registered for the DataType.
EFI_UNSUPPORTED	The notify registration is unsupported, or the specified DataType is not supported.

```
EFI_STATUS EFIAPI EfiIpSecConfigSetData ( IN EFI_IPSEC_CONFIG_PROTOCOL *  This,
                                          IN EFI_IPSEC_CONFIG_DATA_TYPE   DataType,
                                          IN EFI_IPSEC_CONFIG_SELECTOR *  Selector,
                                          IN VOID *                       Data,
                                          IN EFI_IPSEC_CONFIG_SELECTOR *  InsertBefore
                                        )
```

Set the security association, security policy and peer authorization configuration information for the EFI IPsec driver.

This function is used to set the IPsec configuration information of type DataType for the EFI IPsec driver. The IPsec configuration data has a unique selector/identifier separately to identify a data entry. The selector structure depends on DataType's definition. Using SetData() with a Data of NULL causes the IPsec configuration data entry identified by DataType and Selector to be deleted.

Parameters:

[in]	*This*	Pointer to the EFI_IPSEC_CONFIG_PROTOCOL instance.
[in]	*DataType*	The type of data to be set.
[in]	*Selector*	Pointer to an entry selector on operated configuration data specified by DataType. A NULL Selector causes the entire specified-type configuration information to be flushed.
[in]	*Data*	The data buffer to be set. The structure of the data buffer is associated with the DataType.
[in]	*InsertBefore*	Pointer to one entry selector which describes the expected position the new data entry will be added. If InsertBefore is NULL, the new entry will be appended the end of database.

Return values:

EFI_SUCCESS	The specified configuration entry data was set successfully.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: ◊ This is NULL.
EFI_UNSUPPORTED	The specified DataType is not supported.
EFI_OUT_OF_RESOURCED	The required system resource could not be allocated.

Set the security association, security policy and peer authorization configuration information for the EFI IPsec driver.

This function is used to set the IPsec configuration information of type DataType for the EFI IPsec driver. The IPsec configuration data has a unique selector/identifier separately to identify a data entry. The selector structure depends on DataType's definition. Using SetData() with a Data of NULL causes the IPsec configuration data entry identified by DataType and Selector to be deleted.

Parameters:

[in]	*This*	Pointer to the EFI_IPSEC_CONFIG_PROTOCOL instance.
[in]	*DataType*	The type of data to be set.
[in]	*Selector*	Pointer to an entry selector on operated configuration data specified by DataType. A NULL Selector causes the entire specified-type configuration information to be flushed.
[in]	*Data*	The data buffer to be set. The structure of the data buffer is associated with the DataType.
[in]	*InsertBefore*	Pointer to one entry selector which describes the expected position the new data entry will be added. If InsertBefore is NULL, the new entry will be appended to the end of the database.

Return values:

EFI_SUCCESS	The specified configuration entry data was set successfully.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: ◊ This is NULL.
EFI_UNSUPPORTED	The specified DataType is not supported.
EFI_OUT_OF_RESOURCED	The required system resource could not be allocated.

References IpSecConfigSave(), and mSetBySelf.

Referenced by Ikev2ChildSaSilentDelete(), Ikev2StoreSaData(), and IpSecConfigRestore().

```
EFI_STATUS EFIAPI
EfiIpSecConfigUnregisterNotify
(
  IN EFI_IPSEC_CONFIG_PROTOCOL *   This,
  IN EFI_IPSEC_CONFIG_DATA_TYPE    DataType,
  IN EFI_EVENT                     Event
)
```

Remove the specified event that was previously registered on the specified IPsec configuration data.

This function is not supported now and always returns EFI_UNSUPPORTED.

Parameters:

[in]	*This*	Pointer to the EFI_IPSEC_CONFIG_PROTOCOL instance.
[in]	*DataType*	The configuration data type to remove the registered event for.
[in]	*Event*	The event to be unregistered.

Return values:

EFI_SUCCESS	The event was removed successfully.
EFI_NOT_FOUND	The Event specified by DataType could not be found in the database.
EFI_INVALID_PARAMETER	This is NULL or Event is NULL.
EFI_UNSUPPORTED	The notify registration unsupported or the specified DataType is not supported.

Remove the specified event that was previously registered on the specified IPsec configuration data.

This function is not support now and alwasy return EFI_UNSUPPORTED.

Parameters:

[in]	*This*	Pointer to the EFI_IPSEC_CONFIG_PROTOCOL instance.
[in]	*DataType*	The configuration data type to remove the registered event for.
[in]	*Event*	The event to be unregistered.

Return values:

EFI_SUCCESS	The event was removed successfully.
EFI_NOT_FOUND	The Event specified by DataType could not be found in the database.
EFI_INVALID_PARAMETER	This is NULL or Event is NULL.
EFI_UNSUPPORTED	The notify registration is unsupported, or the specified DataType is not supported.

```
VOID FixPadEntry ( IN EFI_IPSEC_PAD_ID *          PadId,
                   IN OUT EFI_IPSEC_PAD_DATA *    Data
                 )
```

Fix the value of some members of PAD ID.

This function is called by IpSecCopyPolicyEntry(), which copy the Policy Entry into the Variable. Since some members in PAD ID are pointers, a physical address to relative address conversion is required before copying this PAD into the variable.

Parameters:

[in]	*PadId*	Pointer of PAD ID.
[in,out]	*Data*	Pointer of PAD Data.

Fix the value of some members of PAD ID.

This function is called by IpSecCopyPolicyEntry()which copy the Policy Entry into the Variable. Since some members in PAD ID are pointers, a physical address to relative address conversion is required before copying this PAD into the variable.

Parameters:

[in]	*PadId*	Pointer of PAD ID.
[in,out]	*Data*	Pointer of PAD Data.

References FIX_REF_BUF_ADDR.

```
VOID FixSadEntry ( IN EFI_IPSEC_SA_ID *           SaId,
                   IN OUT EFI_IPSEC_SA_DATA2 *    Data
                 )
```

Fix the value of some members of SA ID.

This function is called by IpSecCopyPolicyEntry(), which copies the Policy Entry into the Variable. Since some members in SA ID are pointers, a physical address to relative address conversion is required before copying this SAD into the variable.

Parameters:

[in]	*SaId*	Pointer of SA ID.
[in,out]	*Data*	Pointer of SA Data.

Fix the value of some members of SA ID.

This function is called by IpSecCopyPolicyEntry()which copy the Policy Entry into the Variable. Since some members in SA ID are pointers, a physical address to relative address conversion is required before copying this SAD into the variable.

Parameters:

[in]	*SaId*	Pointer of SA ID
[in,out]	*Data*	Pointer of SA Data.

References FIX_REF_BUF_ADDR.

```
VOID FixSpdEntry ( IN EFI_IPSEC_SPD_SELECTOR *   Selector,
                   IN OUT EFI_IPSEC_SPD_DATA *   Data
                 )
```

Fix the value of some members of the SPD Selector.

This function is called by IpSecCopyPolicyEntry(), which copies the Policy Entry into the Variable. Since some members in SPD Selector are pointers, a physical address to relative address conversion is required before copying this SPD entry into the variable.

Parameters:

[in]	*Selector*	Pointer of SPD Selector.
[in,out]	*Data*	Pointer of SPD Data.

Fix the value of some members of SPD Selector.

This function is called by IpSecCopyPolicyEntry()which copy the Policy Entry into the Variable. Since some members in SPD Selector are pointers, a physical address to relative address convertion is required before copying this SPD entry into the variable.

Parameters:

[in]	*Selector*	Pointer of SPD Selector.
[in,out]	*Data*	Pointer of SPD Data.

References FIX_REF_BUF_ADDR.

```
EFI_STATUS GetPadEntry ( IN EFI_IPSEC_CONFIG_SELECTOR *   Selector,
                         IN OUT UINTN *                   DataSize,
                         OUT VOID *                       Data
                       )
```

This function looks up the data entry from IPsec PADand returns the configuration value of the specified PAD Entry.

Parameters:

[in]	*Selector*	Pointer to an entry selector that is an identifier of the PAD entry.

[in,out]	*DataSize*	On output the size of data returned in Data.
[out]	*Data*	The buffer to return the contents of the IPsec configuration data. This type of the data buffer is associated with the DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_NOT_FOUND	The configuration data specified by Selector is not found.
EFI_BUFFER_TOO_SMALL	The DataSize is too small for the result. DataSize has been updated with the size needed to complete the request.

This function lookup the data entry from IPsec PAD. Return the configuration value of the specified PAD Entry.

Parameters:

[in]	*Selector*	Pointer to an entry selector which is an identifier of the PAD entry.
[in,out]	*DataSize*	On output the size of data returned in Data.
[out]	*Data*	The buffer to return the contents of the IPsec configuration data. The type of the data buffer is associated with the DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_NOT_FOUND	The configuration data specified by Selector is not found.
EFI_BUFFER_TOO_SMALL	The DataSize is too small for the result. DataSize has been updated with the size needed to complete the request.

References ComparePadId(), _IPSEC_PAD_ENTRY::Data, _IPSEC_PAD_ENTRY::Id, IPSEC_PAD_ENTRY_FROM_LIST, and mConfigData.

```
EFI_STATUS GetSadEntry ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector,
                         IN OUT UINTN *                  DataSize,
                         OUT VOID *                      Data
                       )
```

This function looks up the data entry from IPsec SAD and returns the configuration value of the specified SAD Entry.

Parameters:

[in]	*Selector*	Pointer to an entry selector that is an identifier of the SAD entry.
[in,out]	*DataSize*	On output, the size of data returned in Data.
[out]	*Data*	The buffer to return the contents of the IPsec configuration data. This type of the data buffer is associated with the DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_NOT_FOUND	The configuration data specified by Selector is not found.
EFI_BUFFER_TOO_SMALL	The DataSize is too small for the result. DataSize has been updated with the size needed to complete the request.

This function lookup the data entry from IPsec SAD. Return the configuration value of the specified SAD Entry.

Parameters:

[in]	*Selector*	Pointer to an entry selector which is an identifier of the SAD entry.
[in,out]	*DataSize*	On output, the size of data returned in Data.
[out]	*Data*	The buffer to return the contents of the IPsec configuration data. The type of the data buffer is associated with the DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_NOT_FOUND	The configuration data specified by Selector is not found.
EFI_BUFFER_TOO_SMALL	The DataSize is too small for the result. DataSize has been updated with the size needed to complete the request.

References _IPSEC_SAD_DATA::AlgoInfo, _IPSEC_SAD_DATA::AntiReplayWindowSize, CompareSaId(), _IPSEC_SAD_ENTRY::Data, DuplicateSpdSelector(), _IPSEC_SAD_ENTRY::Id, IPSEC_SAD_ENTRY_FROM_LIST, _IPSEC_SAD_DATA::ManualSet, mConfigData, _IPSEC_SAD_DATA::Mode, _IPSEC_SAD_DATA::PathMTU, _IPSEC_SAD_DATA::SaLifetime, _IPSEC_SAD_DATA::SequenceNumber, SIZE_OF_SPD_SELECTOR, _IPSEC_SAD_DATA::SpdSelector, _IPSEC_SAD_DATA::TunnelDestAddress, and _IPSEC_SAD_DATA::TunnelSourceAddress.

```
EFI_STATUS GetSpdEntry ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector,
                         IN OUT UINTN *                  DataSize,
                         OUT VOID *                      Data
                       )
```

This function looks up the data entry from IPsec SPD, and returns the configuration value of the specified SPD Entry.

Parameters:

[in]	*Selector*	Pointer to an entry selector which is an identifier of the SPD entry.
[in,out]	*DataSize*	On output the size of data returned in Data.
[out]	*Data*	The buffer to return the contents of the IPsec configuration data. The type of the data buffer is associated with the DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_INVALID_PARAMETER	Data is NULL and *DataSize is not zero.
EFI_NOT_FOUND	The configuration data specified by Selector is not found.
EFI_BUFFER_TOO_SMALL	The DataSize is too small for the result. DataSize has been updated with the size needed to complete the request.

This function lookup the data entry from IPsec SPD. Return the configuration value of the specified SPD Entry.

Parameters:

[in]	*Selector*	Pointer to an entry selector which is an identifier of the SPD entry.
[in,out]	*DataSize*	On output the size of data returned in Data.
[out]	*Data*	The buffer to return the contents of the IPsec configuration data. The type of the data buffer is associated with the DataType.

Return values:

EFI_SUCCESS	The specified configuration data was obtained successfully.
EFI_INVALID_PARAMETER	Data is NULL and *DataSize is not zero.
EFI_NOT_FOUND	The configuration data specified by Selector is not found.

EFI_BUFFER_TOO_SMALL The DataSize is too small for the result. DataSize has been updated with the size needed to complete the request.

References _IPSEC_SPD_DATA::Action, CompareSpdSelector(), _IPSEC_SPD_ENTRY::Data, _IPSEC_SAD_ENTRY::Id, IPSEC_SAD_ENTRY_FROM_SPD, IPSEC_SPD_ENTRY_FROM_LIST, IpSecDuplicateProcessPolicy(), IpSecGetSizeOfSpdData(), mConfigData, _IPSEC_SPD_DATA::Name, _IPSEC_SPD_DATA::PackageFlag, _IPSEC_SPD_DATA::ProcessingPolicy, _IPSEC_SPD_DATA::Sas, _IPSEC_SPD_ENTRY::Selector, and _IPSEC_SPD_DATA::TrafficDirection.

EFI_STATUS IpSecConfigInitialize (IN OUT IPSEC_PRIVATE_DATA * *Private*)

Initialize IPsecConfig protocol

Parameters:

`[in,out]` *Private* Pointer to IPSEC_PRIVATE_DATA. After this function finish, the pointer of IPsecConfig Protocol implementation will copy into its IPsecConfig member.

Return values:

EFI_SUCCESS Initialized the IPsecConfig Protocol successfully.
Others Initializing the IPsecConfig Protocol failed.

Install and Initialize IPsecConfig protocol

Parameters:

`[in,out]` *Private* Pointer to IPSEC_PRIVATE_DATA. After this function finish, the pointer of IPsecConfig Protocol implementation will copy into its IPsecConfig member.

Return values:

EFI_SUCCESS Initialized the IPsecConfig Protocol successfully.
Others Initializing the IPsecConfig Protocol failed.

References IpSecConfigRestore(), and mConfigData.

Referenced by IpSecDriverEntryPoint().

EFI_STATUS IpSecConfigSave (VOID)

This function is the subfunction of the EFIIpSecConfigSetData.

This function call IpSecSetVaraible to set the IPsec Configuration into the firmware.

Return values:

EFI_OUT_OF_RESOURCES The required system resource could not be allocated.
EFI_SUCCESS Saved the configration successfully.
Others Other errors were found while obtaining the variable.

This function is the subfunction of EFIIpSecConfigSetData.

This function call IpSecSetVaraible to set the IPsec Configuration into the firmware.

Return values:

EFI_OUT_OF_RESOURCES The required system resource could not be allocated.
EFI_SUCCESS Saved the configration successfully.

Others Other errors were found while obtaining the variable.

References IPSEC_VARIABLE_BUFFER::Capacity, IPSEC_DEFAULT_VARIABLE_SIZE, IPSECCONFIG_VARIABLE_NAME, IpSecCopyPolicyEntry(), IpSecSetVariable(), IpSecVisitConfigData(), IPSEC_VARIABLE_BUFFER::Ptr, and IPSEC_VARIABLE_BUFFER::Size.

Referenced by EfiIpSecConfigSetData().

VOID IpSecDuplicateProcessPolicy (IN EFI_IPSEC_PROCESS_POLICY * *Dst*,
IN EFI_IPSEC_PROCESS_POLICY * *Src*
)

Copy Source Process Policy to the Destination Process Policy.

Parameters:

[in] *Dst* Pointer to the Source Process Policy.
[in] *Src* Pointer to the Destination Process Policy.

Referenced by GetSpdEntry(), and SetSpdEntry().

UINTN IpSecGetSizeOfEfiSpdData (IN EFI_IPSEC_SPD_DATA * *SpdData*)

Calculate the entire size of EFI_IPSEC_SPD_DATA, which includes the buffer size pointed by the pointer members.

Parameters:

[in] *SpdData* Pointer to a specified EFI_IPSEC_SPD_DATA.

Returns:

The entire size of the specified EFI_IPSEC_SPD_DATA.

Calculate the a whole size of EFI_IPSEC_SPD_DATA, which includes the buffer size pointed to by the pointer members.

Parameters:

[in] *SpdData* Pointer to a specified EFI_IPSEC_SPD_DATA.

Returns:

the whole size the specified EFI_IPSEC_SPD_DATA.

Referenced by SetSpdEntry().

UINTN IpSecGetSizeOfSpdData (IN IPSEC_SPD_DATA * *SpdData*)

Calculate the a entire size of IPSEC_SPD_DATA, which includes the buffer size pointed by the pointer members and the buffer size used by Sa List.

Parameters:

[in] *SpdData* Pointer to the specified IPSEC_SPD_DATA.

Returns:

The entire size of IPSEC_SPD_DATA.

Calculate the a whole size of IPSEC_SPD_DATA which includes the buffer size pointed to by the pointer members and the buffer size used by the Sa List.

Parameters:

[in] *SpdData* Pointer to the specified IPSEC_SPD_DATA.

Returns:

the whole size of IPSEC_SPD_DATA.

Referenced by GetSpdEntry().

```
EFI_STATUS IpSecVisitConfigData ( IN EFI_IPSEC_CONFIG_DATA_TYPE  DataType,
                                  IN IPSEC_COPY_POLICY_ENTRY     Routine,
                                  IN VOID *                      Context
                                )
```

Visit all IPsec Configurations of specified Type and call the caller defined interface.

Parameters:

[in] *DataType* The specified IPsec Config Data Type.
[in] *Routine* The function caller defined.
[in] *Context* The data passed to the Routine.

Return values:

EFI_OUT_OF_RESOURCES The required system resource could not be allocated.
EFI_SUCCESS This function complete successfully.

Visit all IPsec Configurations of specified Type and call the caller defined interface.

Parameters:

[in] *DataType* The specified IPsec Config Data Type.
[in] *Routine* The function defined by the caller.
[in] *Context* The data passed to the Routine.

Return values:

EFI_OUT_OF_RESOURCES The required system resource could not be allocated
EFI_SUCCESS This function completed successfully.

References EfiIpSecConfigGetData(), and EfiIpSecConfigGetNextSelector().

Referenced by Ikev2ChildSaAssociateSpdEntry(), and IpSecConfigSave().

```
BOOLEAN IsSubSpdSelector ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector1,
                           IN EFI_IPSEC_CONFIG_SELECTOR *  Selector2
                         )
```

Find if the two SPD Selectors has subordinative.

Compare two SPD Selector by the fields of LocalAddressCount/RemoteAddressCount/ NextLayerProtocol/LocalPort/LocalPortRange/RemotePort/RemotePortRange and the Local Addresses and

remote Addresses.

Parameters:

[in] *Selector1* Pointer of first SPD Selector.
[in] *Selector2* Pointer of second SPD Selector.

Return values:

TRUE The first SPD Selector is subordinate Selector of second SPD Selector.
FALSE The first SPD Selector is not subordinate Selector of second SPD Selector.

References IsInAddressInfoList().

Referenced by IpSecProcess(), and SetSadEntry().

BOOLEAN IsZeroPadId (IN EFI_IPSEC_CONFIG_SELECTOR * *Selector*)

Check if the PAD ID is Zero.

Parameters:

[in] *Selector* Pointer of the PAD ID.

Return values:

TRUE If the PAD ID is Zero.
FALSE If the PAD ID is not Zero.

BOOLEAN IsZeroSaId (IN EFI_IPSEC_CONFIG_SELECTOR * *Selector*)

Check if the SA ID is Zero by its DestAddress.

Parameters:

[in] *Selector* Pointer of the SA ID.

Return values:

TRUE If the SA ID is Zero.
FALSE If the SA ID is not Zero.

BOOLEAN IsZeroSpdSelector (IN EFI_IPSEC_CONFIG_SELECTOR * *Selector*)

Check if the SPD Selector is Zero by its LocalAddressCount and RemoteAddressCount fields.

Parameters:

[in] *Selector* Pointer of the SPD Selector.

Return values:

TRUE If the SPD Selector is Zero.
FALSE If the SPD Selector is not Zero.

EFI_STATUS SetPadEntry (IN EFI_IPSEC_CONFIG_SELECTOR * *Selector*,
IN VOID * *Data*,
IN VOID * *Context*

)

Set the peer authorization configuration information for the EFI IPsec driver.

The IPsec configuration data has a unique selector/identifier separately to identify a data entry.

Parameters:

[in]	*Selector*	Pointer to an entry selector on operated configuration data specified by DataType. A NULL Selector causes the entire specified-type configuration information to be flushed.
[in]	*Data*	The data buffer to be set. The structure of the data buffer should be EFI_IPSEC_PAD_DATA.
[in]	*Context*	Pointer to one entry selector that describes the expected position where the new data entry will be added. If Context is NULL, the new entry will be appended the end of database.

Return values:

EFI_OUT_OF_RESOURCED	The required system resource could not be allocated.
EFI_SUCCESS	The specified configuration data was obtained successfully.

Set the peer authorization configuration information for the EFI IPsec driver.

The IPsec configuration data has a unique selector/identifier separately to identify a data entry.

Parameters:

[in]	*Selector*	Pointer to an entry selector on operated configuration data specified by DataType. A NULL Selector causes the entire specified-type configuration information to be flushed.
[in]	*Data*	The data buffer to be set. The structure of the data buffer should be EFI_IPSEC_PAD_DATA.
[in]	*Context*	Pointer to one entry selector that describes the expected position the new data entry will be added. If Context is NULL, the new entry will be appended the end of database.

Return values:

EFI_OUT_OF_RESOURCES	The required system resources could not be allocated.
EFI_SUCCESS	The specified configuration data was obtained successfully.

References ComparePadId(), _IPSEC_PAD_ENTRY::Data, _IPSEC_PAD_ENTRY::Id, IPSEC_PAD_ENTRY_FROM_LIST, _IPSEC_PAD_ENTRY::List, and mConfigData.

```
EFI_STATUS SetSadEntry ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector,
                         IN VOID *                       Data,
                         IN VOID *                       Context
                       )
```

Set the security association information for the EFI IPsec driver.

The IPsec configuration data has a unique selector/identifier separately to identify a data entry.

Parameters:

[in]	*Selector*	Pointer to an entry selector on operated configuration data specified by DataType. A NULL Selector causes the entire specified-type configuration information to be flushed.

[in] *Data* The data buffer to be set. The structure of the data buffer should be EFI_IPSEC_SA_DATA.

[in] *Context* Pointer to one entry selector which describes the expected position the new data entry will be added. If Context is NULL,the new entry will be appended to the end of database.

Return values:

EFI_OUT_OF_RESOURCED The required system resource could not be allocated.

EFI_SUCCESS The specified configuration data was obtained successfully.

Set the security association information for the EFI IPsec driver.

The IPsec configuration data has a unique selector/identifier separately to identify a data entry.

Parameters:

[in] *Selector* Pointer to an entry selector on operated configuration data specified by DataType. A NULL Selector causes the entire specified-type configuration information to be flushed.

[in] *Data* The data buffer to be set. The structure of the data buffer should be EFI_IPSEC_SA_DATA.

[in] *Context* Pointer to one entry selector which describes the expected position the new data entry will be added. If Context is NULL,the new entry will be appended the end of database.

Return values:

EFI_OUT_OF_RESOURCED The required system resource could not be allocated.

EFI_SUCCESS The specified configuration data was obtained successfully.

References _IPSEC_SPD_DATA::Action, _IPSEC_SAD_DATA::AlgoInfo, _IPSEC_SAD_DATA::AntiReplayBitmap, _IPSEC_SAD_DATA::AntiReplayWindowSize, _IPSEC_SAD_ENTRY::BySpd, CompareSaId(), _IPSEC_SPD_ENTRY::Data, _IPSEC_SAD_ENTRY::Data, DuplicateSpdSelector(), _IPSEC_SAD_DATA::ESNEnabled, _IPSEC_SAD_ENTRY::Id, IPSEC_SAD_ENTRY_FROM_LIST, IPSEC_SPD_ENTRY_FROM_LIST, IsSubSpdSelector(), _IPSEC_SAD_ENTRY::List, _IPSEC_SAD_DATA::ManualSet, mConfigData, _IPSEC_SAD_DATA::Mode, _IPSEC_SAD_DATA::PathMTU, _IPSEC_SAD_DATA::SaLifetime, _IPSEC_SPD_DATA::Sas, _IPSEC_SPD_ENTRY::Selector, _IPSEC_SAD_DATA::SequenceNumber, SIZE_OF_SPD_SELECTOR, _IPSEC_SAD_DATA::SpdEntry, _IPSEC_SAD_DATA::SpdSelector, _IPSEC_SAD_DATA::TunnelDestAddress, and _IPSEC_SAD_DATA::TunnelSourceAddress.

```
EFI_STATUS SetSpdEntry ( IN EFI_IPSEC_CONFIG_SELECTOR *  Selector,
                         IN VOID *                       Data,
                         IN VOID *                       Context
                       )
```

Set the security policy information for the EFI IPsec driver.

The IPsec configuration data has a unique selector/identifier separately to identify a data entry.

Parameters:

[in] *Selector* Pointer to an entry selector on operated configuration data specified by DataType. A NULL Selector causes the entire specified-type configuration information to be flushed.

[in] *Data*

The data buffer to be set. The structure of the data buffer should be EFI_IPSEC_SPD_DATA.

`[in]` *Context* Pointer to one entry selector that describes the expected position the new data entry will be added. If Context is NULL,the new entry will be appended the end of database.

Return values:

EFI_INVALID_PARAMETER One or more of the following are TRUE:
- ◊ Selector is not NULL and its LocalAddress is NULL or its RemoteAddress is NULL.
- ◊ Data is not NULL, its Action is Protected, and its policy is NULL.
- ◊ Data is not NULL and its Action is not protected and its policy is not NULL.
- ◊ The Action of Data is Protected, its policy mode is Tunnel, and its tunnel option is NULL.
- ◊ The Action of Data is protected, its policy mode is not Tunnel, and it tunnel option is not NULL.

EFI_OUT_OF_RESOURCED The required system resource could not be allocated.

EFI_SUCCESS The specified configuration data was obtained successfully.

Set the security policy information for the EFI IPsec driver.

The IPsec configuration data has a unique selector/identifier separately to identify a data entry.

Parameters:

`[in]` *Selector* Pointer to an entry selector on operated configuration data specified by DataType. A NULL Selector causes the entire specified-type configuration information to be flushed.

`[in]` *Data* The data buffer to be set. The structure of the data buffer should be EFI_IPSEC_SPD_DATA.

`[in]` *Context* Pointer to one entry selector that describes the expected position the new data entry will be added. If Context is NULL, the new entry will be appended the end of database.

Return values:

EFI_INVALID_PARAMETER One or more of the following are TRUE:
- ◊ Selector is not NULL and its LocalAddress is NULL or its RemoteAddress is NULL.
- ◊ Data is not NULL and its Action is Protected and its plolicy is NULL.
- ◊ Data is not NULL, its Action is not protected, and its policy is not NULL.
- ◊ The Action of Data is Protected, its policy mode is Tunnel, and its tunnel option is NULL.
- ◊ The Action of Data is protected and its policy mode is not Tunnel and it tunnel option is not NULL.

EFI_OUT_OF_RESOURCED The required system resource could not be allocated.

EFI_SUCCESS The specified configuration data was obtained successfully.

References _IPSEC_SPD_DATA::Action, _IPSEC_SAD_ENTRY::BySpd, CompareSaId(), CompareSpdSelector(), _IPSEC_SAD_ENTRY::Data, _IPSEC_SPD_ENTRY::Data, DuplicateSpdSelector(), _IPSEC_SAD_ENTRY::Id, IPSEC_SAD_ENTRY_FROM_LIST, IPSEC_SAD_ENTRY_FROM_SPD, IPSEC_SPD_ENTRY_FROM_LIST, IpSecDuplicateProcessPolicy(), IpSecGetSizeOfEfiSpdData(),

_IPSEC_SPD_ENTRY::List, mConfigData, _IPSEC_SPD_DATA::Name, _IPSEC_SPD_DATA::PackageFlag, _IPSEC_SPD_DATA::ProcessingPolicy, _IPSEC_SPD_DATA::Sas, _IPSEC_SPD_ENTRY::Selector, SIZE_OF_SPD_SELECTOR, _IPSEC_SAD_DATA::SpdEntry, and _IPSEC_SPD_DATA::TrafficDirection.

```
VOID UnfixPadEntry ( IN EFI_IPSEC_PAD_ID *          PadId,
                     IN OUT EFI_IPSEC_PAD_DATA *    Data
                   )
```

Recover the value of some members of PAD ID.

This function is corresponding to FixPadEntry(). It recovers the value of members of PAD ID which fix by the FixPadEntry().

Parameters:

[in]	*PadId*	Pointer of PAD ID
[in,out]	*Data*	Pointer of PAD Data.

Recover the value of some members of PAD ID.

This function is corresponding to FixPadEntry(). It recovers the value of members of PAD ID that are fixed by FixPadEntry().

Parameters:

[in]	*PadId*	Pointer of PAD ID.
[in,out]	*Data*	Pointer of PAD Data.

References UNFIX_REF_BUF_ADDR.

```
VOID UnfixSadEntry ( IN OUT EFI_IPSEC_SA_ID *          SaId,
                     IN OUT EFI_IPSEC_SA_DATA2 *       Data
                   )
```

Recover the value of some members of SA ID.

This function is corresponding to FixSadEntry(). It recovers the value of members of SAD ID which fix by the FixSadEntry().

Parameters:

[in,out]	*SaId*	Pointer of SAD ID
[in,out]	*Data*	Pointer of SAD Data.

Recover the value of some members of SA ID.

This function is corresponding to FixSadEntry(). It recovers the value of members of SAD ID that are fixed by FixSadEntry().

Parameters:

[in,out]	*SaId*	Pointer of SAD ID.
[in,out]	*Data*	Pointer of SAD Data.

References UNFIX_REF_BUF_ADDR.

```
VOID UnfixSpdEntry ( IN OUT EFI_IPSEC_SPD_SELECTOR *   Selector,
                     IN OUT EFI_IPSEC_SPD_DATA *       Data
                   )
```

Recover the value of some members of SPD Selector.

This function is corresponding to FixSpdEntry(). It recovers the value of members of SPD Selector which fix by the FixSpdEntry().

Parameters:

[in,out]	*Selector*	Pointer of SPD Selector.
[in,out]	*Data*	Pointer of SPD Data.

Recover the value of some members of SPD Selector.

This function is corresponding to FixSpdEntry(). It recovers the value of members of SPD Selector that are fixed by FixSpdEntry().

Parameters:

[in,out]	*Selector*	Pointer of SPD Selector.
[in,out]	*Data*	Pointer of SPD Data.

References UNFIX_REF_BUF_ADDR.

Variable Documentation

LIST_ENTRY mConfigData[IPsecConfigDataTypeMaximum]

Referenced by EfiIpSecConfigGetNextSelector(), GetPadEntry(), GetSadEntry(), GetSpdEntry(), IkeSearchSpdEntry(), IpSecConfigInitialize(), IpSecLookupPadEntry(), IpSecLookupSadBySpi(), IpSecProcess(), SetPadEntry(), SetSadEntry(), and SetSpdEntry().

NetworkPkg/IpSecDxe/IpSecCryptIo.c File Reference

Functions

UINTN IpSecGetEncryptBlockSize (IN UINT8 AlgorithmId)
UINTN IpSecGetEncryptKeyLength (IN UINT8 AlgorithmId)
UINTN IpSecGetEncryptIvLength (IN UINT8 AlgorithmId)
UINTN IpSecGetHmacDigestLength (IN UINT8 AlgorithmId)
UINTN IpSecGetIcvLength (IN UINT8 AlgorithmId)
EFI_STATUS IpSecGenerateIv (IN UINT8 *IvBuffer, IN UINTN IvSize)
UINTN IpSecGetIndexFromEncList (IN UINT8 AlgorithmId)
UINTN IpSecGetIndexFromAuthList (IN UINT8 AlgorithmId)
EFI_STATUS IpSecCryptoIoEncrypt (IN CONST UINT8 AlgorithmId, IN CONST UINT8 *Key, IN CONST UINTN KeyBits, IN CONST UINT8 *Ivec, IN UINT8 *InData, IN UINTN InDataLength, OUT UINT8 *OutData)
EFI_STATUS IpSecCryptoIoDecrypt (IN CONST UINT8 AlgorithmId, IN CONST UINT8 *Key, IN CONST UINTN KeyBits, IN CONST UINT8 *Ivec, IN UINT8 *InData, IN UINTN InDataLength, OUT UINT8 *OutData)
EFI_STATUS IpSecCryptoIoHmac (IN CONST UINT8 AlgorithmId, IN CONST UINT8 *Key, IN UINTN KeyLength, IN HASH_DATA_FRAGMENT *InDataFragment, IN UINTN FragmentCount, OUT UINT8 *OutData, IN UINTN OutDataSize)
EFI_STATUS IpSecCryptoIoHash (IN CONST UINT8 AlgorithmId, IN HASH_DATA_FRAGMENT *InDataFragment, IN UINTN FragmentCount, OUT UINT8 *OutData, IN UINTN OutDataSize)
EFI_STATUS IpSecCryptoIoDhGetPublicKey (IN OUT UINT8 **DhContext, IN UINTN Generator, IN UINTN PrimeLength, IN CONST UINT8 *Prime, OUT UINT8 *PublicKey, IN OUT UINTN *PublicKeySize)
EFI_STATUS IpSecCryptoIoDhComputeKey (IN OUT UINT8 *DhContext, IN CONST UINT8 *PeerPublicKey, IN UINTN PeerPublicKeySize, OUT UINT8 *Key, IN OUT UINTN *KeySize)
EFI_STATUS IpSecCryptoIoFreeDh (IN OUT UINT8 **DhContext)
EFI_STATUS IpSecCryptoIoGenerateRandomBytes (OUT UINT8 *OutBuffer, IN UINTN Bytes)
VOID IpSecCryptoIoAuthDataWithCertificate (IN UINT8 *InData, IN UINTN InDataSize, IN UINT8 *PrivateKey, IN UINTN PrivateKeySize, IN UINT8 *KeyPassWord, IN UINTN KeyPwdSize, OUT UINT8 **OutData, IN OUT UINTN *OutDataSize)
BOOLEAN IpSecCryptoIoVerifySignDataByCertificate (IN UINT8 *InCert, IN UINTN CertLen, IN UINT8 *InCa, IN UINTN CaLen, IN UINT8 *InData, IN UINTN InDataSize, IN UINT8 *Singnature, IN UINTN SigSize)
EFI_STATUS

IpSecCryptoIoGetPublicKeyFromCert (IN UINT8 *InCert, IN UINTN CertLen, OUT UINT8 **PublicKey, OUT UINTN *PublicKeyLen)

EFI_STATUS IpSecCryptoIoGetSubjectFromCert (IN UINT8 *InCert, IN UINTN CertSize, OUT UINT8 **CertSubject, OUT UINTN *SubjectSize)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED mIpsecEncryptAlgorithmList
ENCRYPT_ALGORITHM [IPSEC_ENCRYPT_ALGORITHM_LIST_SIZE]

GLOBAL_REMOVE_IF_UNREFERENCED mIpsecAuthAlgorithmList
AUTH_ALGORITHM [IPSEC_AUTH_ALGORITHM_LIST_SIZE]

GLOBAL_REMOVE_IF_UNREFERENCED mIpsecHashAlgorithmList
HASH_ALGORITHM [IPSEC_HASH_ALGORITHM_LIST_SIZE]

BOOLEAN mInitialRandomSeed = FALSE

Detailed Description

Common interfaces to call Security library.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
VOID IpSecCryptoIoAuthDataWithCertificate ( IN UINT8 *          InData,
                                            IN UINTN            InDataSize,
                                            IN UINT8 *          PrivateKey,
                                            IN UINTN            PrivateKeySize,
                                            IN UINT8 *          KeyPassWord,
                                            IN UINTN            KeyPwdSize,
                                            OUT UINT8 **        OutData,
                                            IN OUT UINTN *      OutDataSize
                                          )
```

Authenticate data with the certificate.

Parameters:

[in]	*InData*	Pointer to the Data to be signed.
[in]	*InDataSize*	InData size in bytes.
[in]	*PrivateKey*	Pointer to the private key.
[in]	*PrivateKeySize*	The size of Private Key in bytes.

[in]	*KeyPassWord*	Pointer to the password for retrieving private key.
[in]	*KeyPwdSize*	The size of Key Password in bytes.
[out]	*OutData*	The pointer to the signed data.
[in,out]	*OutDataSize*	Pointer to contain the size of out data.

Referenced by Ikev2CertGenerateAuthPayload().

```
EFI_STATUS IpSecCryptoIoDecrypt (
    IN CONST UINT8 AlgorithmId,
    IN CONST UINT8 * Key,
    IN CONST UINTN KeyBits,
    IN CONST UINT8 * Ivec,
    IN UINT8 * InData,
    IN UINTN InDataLength,
    OUT UINT8 * OutData
)
```

Decrypts the buffer.

This function calls relevant Decryption interface from CryptoLib according to the input alogrithm ID. The InData should be multiple of block size. This function doesn't perform the padding. If it has the Ivec data, the length of it should be same with the block size. The block size is different from the different algorithm.

Parameters:

[in]	*AlgorithmId*	The Alogrithem identification defined in RFC.
[in]	*Key*	Pointer to the buffer containing encrypting key.
[in]	*KeyBits*	The length of the key in bits.
[in]	*Ivec*	Point to the buffer containning the Initializeion Vector (IV) data.
[in]	*InData*	Point to the buffer containing the data to be decrypted.
[in]	*InDataLength*	The length of InData in Bytes.
[out]	*OutData*	Pointer to the buffer that receives the decryption output.

Return values:

EFI_UNSUPPORTED	The input Algorithm is not supported.
EFI_OUT_OF_RESOURCE	The required resource can't be allocated.
EFI_SUCCESS	The operation completed successfully.

References _ENCRYPT_ALGORITHM::CipherGetContextSize, IKE_EALG_3DESCBC, IKE_EALG_AESCBC, IKE_EALG_NONE, IKE_EALG_NULL, IpSecGetIndexFromEncList(), and mIpsecEncryptAlgorithmList.

Referenced by Ikev2DecryptPacket(), and IpSecEspInboundPacket().

```
EFI_STATUS IpSecCryptoIoDhComputeKey (
    IN OUT UINT8 * DhContext,
```

```
                                IN CONST UINT8 *   PeerPublicKey,
                                IN UINTN           PeerPublicKeySize,
                                OUT UINT8 *        Key,
                                IN OUT UINTN *     KeySize
                              )
```

Generates exchanged common key.

Given peer's public key, this function computes the exchanged common key, based on its own context including value of prime modulus and random secret exponent.

Parameters:

`[in,out]`	*DhContext*	Pointer to the DH context.
`[in]`	*PeerPublicKey*	Pointer to the peer's Public Key.
`[in]`	*PeerPublicKeySize*	Size of peer's public key in bytes.
`[out]`	*Key*	Pointer to the buffer to receive generated key.
`[in,out]`	*KeySize*	For in, the size of Key buffer in bytes. For out, the size of data returned in Key buffer in bytes.

Return values:

EFI_SUCCESS	The operation perfoms successfully.
Otherwise	The operation is failed.

Referenced by Ikev2GenerateSaDhComputeKey().

```
EFI_STATUS IpSecCryptoIoDhGetPublicKey ( IN OUT UINT8 **      DhContext,
                                         IN UINTN             Generator,
                                         IN UINTN             PrimeLength,
                                         IN CONST UINT8 *     Prime,
                                         OUT UINT8 *          PublicKey,
                                         IN OUT UINTN *       PublicKeySize
                                       )
```

Generates the Diffie-Hellman public key.

This function first initiate a DHContext, then call the DhSetParameter() to set the prime and primelenght, at end call the DhGenerateKey() to generates random secret exponent, and computes the public key. The output returned via parameter PublicKey and PublicKeySize. DH context is updated accordingly. If the PublicKey buffer is too small to hold the public key, EFI_INVALID_PARAMETER is returned and PublicKeySize is set to the required buffer size to obtain the public key.

Parameters:

`[in,out]`	*DhContext*	Pointer to the DH context.
`[in]`	*Generator*	Vlaue of generator.
`[in]`	*PrimeLength*	Length in bits of prime to be generated.
`[in]`	*Prime*	Pointer to the buffer to receive the generated prime number.
`[out]`	*PublicKey*	Pointer to the buffer to receive generated public key.
`[in,out]`	*PublicKeySize*	For in, the size of PublicKey buffer in bytes. For out, the size of data returned in PublicKey buffer in bytes.

Return values:

EFI_SUCCESS	The operation perfoms successfully.
Otherwise	The operation is failed.

Referenced by Ikev2GenerateSaDhPublicKey().

```
EFI_STATUS IpSecCryptoIoEncrypt ( IN CONST UINT8      AlgorithmId,
                                  IN CONST UINT8 *    Key,
                                  IN CONST UINTN      KeyBits,
                                  IN CONST UINT8 *    Ivec,
                                  IN UINT8 *          InData,
                                  IN UINTN            InDataLength,
                                  OUT UINT8 *         OutData
                                )
```

Encrypt the buffer.

This function calls relevant encryption interface from CryptoLib according to the input alogrithm ID. The InData should be multiple of block size. This function doesn't perform the padding. If it has the Ivec data, the length of it should be same with the block size. The block size is different from the different algorithm.

Parameters:

[in]	*AlgorithmId*	The Alogrithem identification defined in RFC.
[in]	*Key*	Pointer to the buffer containing encrypting key.
[in]	*KeyBits*	The length of the key in bits.
[in]	*Ivec*	Point to the buffer containning the Initializeion Vector (IV) data.
[in]	*InData*	Point to the buffer containing the data to be encrypted.
[in]	*InDataLength*	The length of InData in Bytes.
[out]	*OutData*	Point to the buffer that receives the encryption output.

Return values:

EFI_UNSUPPORTED	The input Algorithm is not supported.
EFI_OUT_OF_RESOURCE	The required resource can't be allocated.
EFI_SUCCESS	The operation completed successfully.

References _ENCRYPT_ALGORITHM::CipherGetContextSize, IKE_EALG_3DESCBC, IKE_EALG_AESCBC, IKE_EALG_NONE, IKE_EALG_NULL, IpSecGetIndexFromEncList(), and mIpsecEncryptAlgorithmList.

Referenced by Ikev2EncryptPacket(), and IpSecEspOutboundPacket().

```
EFI_STATUS IpSecCryptoIoFreeDh ( IN OUT UINT8 **   DhContext   )
```

Releases the DH context. If DhContext is NULL, return EFI_INVALID_PARAMETER.

Parameters:

[in,out]	*DhContext*	Pointer to the DH context to be freed.

Return values:

EFI_SUCCESS	The operation perfoms successfully.
EFI_INVALID_PARAMETER	The DhContext is NULL.

Referenced by Ikev2DhBufferFree().

```
EFI_STATUS IpSecCryptoIoGenerateRandomBytes ( OUT UINT8 *   OutBuffer,
                                              IN UINTN      Bytes
                                            )
```

Generates random numbers of specified size.

If the Random Generator wasn't initiated, initiate it first, then call RandomBytes.

Parameters:

[out]	*OutBuffer*	Pointer to buffer to receive random value.
[in]	*Bytes*	Size of randome bytes to generate.

Return values:

EFI_SUCCESS	The operation perfoms successfully.
Otherwise	The operation is failed.

References mInitialRandomSeed.

Referenced by IkeGenerateCookie(), IkeGenerateIv(), IkeGenerateNonce(), and IpSecGenerateIv().

```
EFI_STATUS IpSecCryptoIoGetPublicKeyFromCert ( IN UINT8 *      InCert,
                                               IN UINTN        CertLen,
                                               OUT UINT8 **    PublicKey,
                                               OUT UINTN *     PublicKeyLen
                                             )
```

Retrieves the RSA Public Key from one X509 certificate (DER format only).

Parameters:

[in]	*InCert*	Pointer to the certificate.
[in]	*CertLen*	The size of the certificate in bytes.
[out]	*PublicKey*	Pointer to the retrieved public key.
[out]	*PublicKeyLen*	Size of Public Key in bytes.

Return values:

EFI_SUCCESS	Successfully get the public Key.
EFI_INVALID_PARAMETER	The certificate is malformed.

Referenced by Ikev2GenerateCertificatePayload().

```
EFI_STATUS IpSecCryptoIoGetSubjectFromCert ( IN UINT8 *      InCert,
                                             IN UINTN        CertSize,
                                             OUT UINT8 **    CertSubject,
                                             OUT UINTN *     SubjectSize
                                           )
```

Retrieves the subject name from one X509 certificate (DER format only).

Parameters:

[in]	*InCert*	Pointer to the X509 certificate.
[in]	*CertSize*	The size of the X509 certificate in bytes.
[out]	*CertSubject*	Pointer to the retrieved certificate subject.
[out]	*SubjectSize*	The size of Certificate Subject in bytes.

Return values:

EFI_SUCCESS	Retrieved the certificate subject successfully.
EFI_INVALID_PARAMETER	The certificate is malformed.

Referenced by Ikev2GenerateCertIdPayload().

```
EFI_STATUS IpSecCryptoIoHash ( IN CONST UINT8                AlgorithmId,
                               IN HASH_DATA_FRAGMENT *       InDataFragment,
                               IN UINTN                      FragmentCount,
                               OUT UINT8 *                   OutData,
                               IN UINTN                      OutDataSize
                             )
```

Digests the Payload and store the result into the OutData.

This function calls relevant Hash interface from CryptoLib according to the input alogrithm ID. It computes all datas from InDataFragment and output the result into the OutData buffer. If the OutDataSize is larger than the related Hash alogrithm output size, return EFI_INVALID_PARAMETER.

Parameters:

[in]	*AlgorithmId*	The authentication Identification.
[in]	*InDataFragment*	A list contains all data to be authenticated.
[in]	*FragmentCount*	The size of the InDataFragment.
[out]	*OutData*	For in, the buffer to receive the output data. For out, the buffer contains the authenticated data.
[in]	*OutDataSize*	The size of the buffer of OutData.

Return values:

EFI_UNSUPPORTED	If the AuthAlg is not in the support list.
EFI_SUCCESS	Authenticated the payload successfully.
EFI_INVALID_PARAMETER	If the OutDataSize is larger than the related Hash algorithm could handle.
otherwise	Authentication of the payload failed.

References _HASH_ALGORITHM::HashGetContextSize, IKE_AALG_NONE, IKE_AALG_NULL, IKE_AALG_SHA1HMAC, IpSecGetHmacDigestLength(), IpSecGetIndexFromAuthList(), and mIpsecHashAlgorithmList.

Referenced by Ikev2CertGenerateAuthPayload(), and Ikev2GenerateCertificatePayload().

```
EFI_STATUS IpSecCryptoIoHmac ( IN CONST UINT8                AlgorithmId,
                               IN CONST UINT8 *              Key,
                               IN UINTN                      KeyLength,
                               IN HASH_DATA_FRAGMENT *       InDataFragment,
                               IN UINTN                      FragmentCount,
```

```
                                OUT UINT8 *          OutData,
                                IN UINTN             OutDataSize
                              )
```

Digests the Payload with key and store the result into the OutData.

This function calls relevant Hmac interface from CryptoLib according to the input alogrithm ID. It computes all datas from InDataFragment and output the result into the OutData buffer. If the OutDataSize is larger than the related HMAC alogrithm output size, return EFI_INVALID_PARAMETER.

Parameters:

[in]	*AlgorithmId*	The authentication Identification.
[in]	*Key*	Pointer of the authentication key.
[in]	*KeyLength*	The length of the Key in bytes.
[in]	*InDataFragment*	The list contains all data to be authenticated.
[in]	*FragmentCount*	The size of the InDataFragment.
[out]	*OutData*	For in, the buffer to receive the output data. For out, the buffer contains the authenticated data.
[in]	*OutDataSize*	The size of the buffer of OutData.

Return values:

EFI_UNSUPPORTED	If the AuthAlg is not in the support list.
EFI_INVALID_PARAMETER	The OutData buffer size is larger than algorithm digest size.
EFI_SUCCESS	Authenticate the payload successfully.
otherwise	Authentication of the payload fails.

References _AUTH_ALGORITHM::HmacGetContextSize, IKE_AALG_NONE, IKE_AALG_NULL, IKE_AALG_SHA1HMAC, IpSecGetHmacDigestLength(), IpSecGetIndexFromAuthList(), and mIpsecAuthAlgorithmList.

Referenced by Ikev2CertGenerateAuthPayload(), Ikev2DecryptPacket(), Ikev2EncryptPacket(), Ikev2GenerateSaKeys(), Ikev2PskGenerateAuthPayload(), Ikev2SaGenerateKey(), IpSecEspAuthVerifyPayload(), and IpSecEspOutboundPacket().

```
BOOLEAN IpSecCryptoIoVerifySignDataByCertificate ( IN UINT8 *   InCert,
                                                   IN UINTN     CertLen,
                                                   IN UINT8 *   InCa,
                                                   IN UINTN     CaLen,
                                                   IN UINT8 *   InData,
                                                   IN UINTN     InDataSize,
                                                   IN UINT8 *   Singnature,
                                                   IN UINTN     SigSize
                                                 )
```

Verify the singed data with the public key which is contained in a certificate.

Parameters:

[in]	*InCert*	Pointer to the Certificate which contains the public key.
[in]	*CertLen*	The size of Certificate in bytes.
[in]	*InCa*	Pointer to the CA certificate
[in]	*CaLen*	The size of CA certificate in bytes.

[in] *InData* Pointer to octect message hash to be checked.
[in] *InDataSize* Size of the message hash in bytes.
[in] *Singnature* The pointer to the RSA PKCS1-V1_5 signature to be verifed.
[in] *SigSize* Size of signature in bytes.

Return values:
TRUE Valid signature encoded in PKCS1-v1_5.
FALSE Invalid signature or invalid RSA context.

Referenced by Ikev2AuthCertParser().

EFI_STATUS IpSecGenerateIv (IN UINT8 * *IvBuffer*,
IN UINTN *IvSize*
)

Generate a random data for IV. If the IvSize is zero, not needed to create IV and return EFI_SUCCESS.

Parameters:
[in] *IvBuffer* The pointer of the IV buffer.
[in] *IvSize* The IV size in bytes.

Return values:
EFI_SUCCESS Create a random data for IV.

References IpSecCryptoIoGenerateRandomBytes().

Referenced by IpSecEspOutboundPacket().

UINTN IpSecGetEncryptBlockSize (IN UINT8 *AlgorithmId*)

Get the block size of specified encryption alogrithm.

Parameters:
[in] *AlgorithmId* The encryption algorithm ID.

Returns:
The value of block size.

References _ENCRYPT_ALGORITHM::BlockSize, IPSEC_ENCRYPT_ALGORITHM_LIST_SIZE, and mIpsecEncryptAlgorithmList.

Referenced by Ikev2DecryptPacket(), Ikev2EncryptPacket(), IpSecEspInboundPacket(), and IpSecEspOutboundPacket().

UINTN IpSecGetEncryptIvLength (IN UINT8 *AlgorithmId*)

Get the IV size of the specified encryption alogrithm.

Parameters:
[in] *AlgorithmId* The encryption algorithm ID.

Returns:

The value of IV size.

References IPSEC_ENCRYPT_ALGORITHM_LIST_SIZE, _ENCRYPT_ALGORITHM::IvLength, and mIpsecEncryptAlgorithmList.

Referenced by IpSecEspInboundPacket(), and IpSecEspOutboundPacket().

UINTN IpSecGetEncryptKeyLength (IN UINT8 *AlgorithmId*)
Get the key length of the specified encryption alogrithm.

Parameters:
[in] *AlgorithmId* The encryption algorithm ID.

Returns:
The value of key length.

References IPSEC_ENCRYPT_ALGORITHM_LIST_SIZE, _ENCRYPT_ALGORITHM::KeyLength, and mIpsecEncryptAlgorithmList.

Referenced by Ikev2GenerateChildSaKeys(), Ikev2GenerateSaKeys(), Ikev2InitializeSaData(), and Ikev2ParseProposalData().

UINTN IpSecGetHmacDigestLength (IN UINT8 *AlgorithmId*)
Get the HMAC digest length by the specified Algorithm ID.

Parameters:
[in] *AlgorithmId* The specified Alogrithm ID.

Returns:
The digest length of the specified Authentication Algorithm ID.

References _AUTH_ALGORITHM::DigestLength, IPSEC_AUTH_ALGORITHM_LIST_SIZE, and mIpsecAuthAlgorithmList.

Referenced by Ikev2CertGenerateAuthPayload(), Ikev2GenerateCertificatePayload(), Ikev2GenerateChildSaKeys(), Ikev2GenerateSaKeys(), Ikev2PskGenerateAuthPayload(), Ikev2SaGenerateKey(), IpSecCryptoIoHash(), and IpSecCryptoIoHmac().

UINTN IpSecGetIcvLength (IN UINT8 *AlgorithmId*)
Get the ICV size of the specified Authenticaion alogrithm.

Parameters:
[in] *AlgorithmId* The Authentication algorithm ID.

Returns:
The value of ICV size.

References _AUTH_ALGORITHM::IcvLength, IPSEC_AUTH_ALGORITHM_LIST_SIZE, and mIpsecAuthAlgorithmList.

Referenced by Ikev2DecryptPacket(), Ikev2EncryptPacket(), IpSecEspInboundPacket(), and IpSecEspOutboundPacket().

UINTN IpSecGetIndexFromAuthList (IN UINT8 *AlgorithmId*)

Get index of the specified encryption alogrithm from the mIpsecAuthAlgorithemList.

Parameters:

`[in]` *AlgorithmId* The encryption algorithm ID.

Returns:

the index.

References IPSEC_AUTH_ALGORITHM_LIST_SIZE, and mIpsecAuthAlgorithmList.

Referenced by IpSecCryptoIoHash(), and IpSecCryptoIoHmac().

UINTN IpSecGetIndexFromEncList (IN UINT8 *AlgorithmId*)

Get index of the specified encryption alogrithm from the mIpsecEncryptAlgorithemList.

Parameters:

`[in]` *AlgorithmId* The encryption algorithm ID.

Returns:

the index.

References IPSEC_ENCRYPT_ALGORITHM_LIST_SIZE, and mIpsecEncryptAlgorithmList.

Referenced by IpSecCryptoIoDecrypt(), and IpSecCryptoIoEncrypt().

Variable Documentation

BOOLEAN mInitialRandomSeed = FALSE

Referenced by IpSecCryptoIoGenerateRandomBytes().

GLOBAL_REMOVE_IF_UNREFERENCED AUTH_ALGORITHM mIpsecAuthAlgorithmList[IPSEC_AUTH_ALGORITHM_LIST_SIZE]

Initial value:

```
{
  {IKE_AALG_NONE, 0, 0, 0, NULL, NULL, NULL, NULL},
  {IKE_AALG_NULL, 0, 0, 0, NULL, NULL, NULL, NULL},
  {IKE_AALG_SHA1HMAC, 20, 12, 64, HmacSha1GetContextSize, HmacSha1Init, HmacSha1Update, HmacSha1F
}
```

Referenced by IpSecCryptoIoHmac(), IpSecGetHmacDigestLength(), IpSecGetIcvLength(), and IpSecGetIndexFromAuthList().

GLOBAL_REMOVE_IF_UNREFERENCED ENCRYPT_ALGORITHM mIpsecEncryptAlgorithmList[IPSEC_ENCRYPT_ALGORITHM_LIST_SIZE]

Initial value:

```
 {
  {IKE_EALG_NULL, 0, 0, 1, NULL, NULL, NULL, NULL},
  {IKE_EALG_NONE, 0, 0, 1, NULL, NULL, NULL, NULL},
  {IKE_EALG_3DESCBC, 24, 8, 8, TdesGetContextSize, TdesInit, TdesCbcEncrypt, TdesCbcDecrypt},
  {IKE_EALG_AESCBC, 16, 16, 16, AesGetContextSize, AesInit, AesCbcEncrypt, AesCbcDecrypt}
}
```

Referenced by IpSecCryptoIoDecrypt(), IpSecCryptoIoEncrypt(), IpSecGetEncryptBlockSize(), IpSecGetEncryptIvLength(), IpSecGetEncryptKeyLength(), and IpSecGetIndexFromEncList().

GLOBAL_REMOVE_IF_UNREFERENCED HASH_ALGORITHM mIpsecHashAlgorithmList[IPSEC_HASH_ALGORITHM_LIST_SIZE]

Initial value:

```
 {
  {IKE_AALG_NONE, 0, 0, 0, NULL, NULL, NULL, NULL},
  {IKE_AALG_NULL, 0, 0, 0, NULL, NULL, NULL, NULL},
  {IKE_AALG_SHA1HMAC, 20, 12, 64, Sha1GetContextSize, Sha1Init, Sha1Update, Sha1Final}
}
```

Referenced by IpSecCryptoIoHash().

NetworkPkg/IpSecDxe/IpSecCryptIo.h File Reference

Data Structures

struct _ENCRYPT_ALGORITHM
struct _AUTH_ALGORITHM
struct _HASH_ALGORITHM

Defines

"memItemRight" valign="bottom">IPSEC_ENCRYPT_ALGORITHM_LIST_SIZE 4
#define IPSEC_AUTH_ALGORITHM
#define IPSEC_HASH_ALGORITHM
#define IKE_AALG_NONE 0x00
#define IKE_AALG_SHA1HMAC 0
#define IKE_AALG_NULL 0xFB
#define IKE_EALG_NONE 0x00
#define IKE_EALG_3DESCBC 0x0
#define IKE_EALG_NULL 0x0B
#define IKE_EALG_AESCBC 0x0C

Typedefs

typedef UINTN(EFIAPI * CRYPTO_HMAC_GETCON
)(VOID)
typedef BOOLEAN(EFIAPI * CRYPTO_HMAC_INIT)(OU
*Context, IN CONST UINT8
UINTN KeySize)
typedef BOOLEAN(EFIAPI * CRYPTO_HMAC_UPDATE
*Context, IN CONST VOID *
DataLength)
typedef BOOLEAN(EFIAPI * CRYPTO_HMAC_FINAL)(I
*Context, OUT UINT8 *Hma
typedef UINTN(EFIAPI * CRYPTO_CIPHER_GETCO
)(VOID)
typedef BOOLEAN(EFIAPI * CRYPTO_CIPHER_INIT)(IN
*Context, IN CONST UINT8
UINTN KeyBits)
typedef BOOLEAN(EFIAPI * CRYPTO_CIPHER_ENCRY
*Context, IN CONST UINT8
UINTN InputSize, IN CONST
OUT UINT8 *OutData)
typedef BOOLEAN(EFIAPI * CRYPTO_CIPHER_DECRY
*Context, IN CONST UINT8
UINTN InputSize, IN CONST
OUT UINT8 *OutData)
typedef UINTN(EFIAPI * CRYPTO_HASH_GETCONT
)(VOID)
typedef BOOLEAN(EFIAPI *

CRYPTO_HASH_INIT)(OUT VOID *Context)

typedef BOOLEAN(EFIAPI * CRYPTO_HASH_UPDATE)(IN OU *Context, IN CONST VOID *Data, IN DataSize)

typedef BOOLEAN(EFIAPI * CRYPTO_HASH_FINAL)(IN OUT *Context, OUT UINT8 *HashValue)

typedef struct _ENCRYPT_ALGORITHM ENCRYPT_ALGORITHM

typedef struct _AUTH_ALGORITHM AUTH_ALGORITHM

typedef struct _HASH_ALGORITHM HASH_ALGORITHM

Functions

UINTN IpSecGetEncryptIvLength (IN UINT8 AlgorithmId)

UINTN IpSecGetEncryptBlockSize (IN UINT AlgorithmId)

UINTN IpSecGetEncryptKeyLength (IN UINT AlgorithmId)

UINTN IpSecGetIcvLength (IN UINT8 Algori

UINTN IpSecGetHmacDigestLength (IN UIN AlgorithmId)

EFI_STATUS IpSecGenerateIv (IN UINT8 *IvBuffe UINTN IvSize)

EFI_STATUS IpSecCryptoIoEncrypt (IN CONST U AlgorithmId, IN CONST UINT8 *Key CONST UINTN KeyBits, IN CONST *Ivec, IN UINT8 *InData, IN UINTN InDataLength, OUT UINT8 *OutData

EFI_STATUS IpSecCryptoIoDecrypt (IN CONST U AlgorithmId, IN CONST UINT8 *Key CONST UINTN KeyBits, IN CONST *Ivec, IN UINT8 *InData, IN UINTN InDataLength, OUT UINT8 *OutData

EFI_STATUS IpSecCryptoIoHmac (IN CONST UIN AlgorithmId, IN CONST UINT8 *Key UINTN KeyLength, IN HASH_DATA_FRAGMENT *InDataFragment, IN UINTN Fragme OUT UINT8 *OutData, IN UINTN OutDataSize)

EFI_STATUS IpSecCryptoIoHash (IN CONST UIN AlgorithmId, IN HASH_DATA_FRA *InDataFragment, IN UINTN Fragme OUT UINT8 *OutData, IN UINTN OutDataSize)

EFI_STATUS IpSecCryptoIoDhGetPublicKey (IN O UINT8 **DhContext, IN UINTN Gen IN UINTN PrimeLength, IN CONST *Prime, OUT UINT8 *PublicKey, IN UINTN *PublicKeySize)

EFI_STATUS IpSecCryptoIoDhComputeKey (IN OU UINT8 *DhContext, IN CONST UIN *PeerPublicKey, IN UINTN

PeerPublicKeySize, OUT UIN
OUT UINTN *KeySize)
EFI_STATUS IpSecCryptoIoFreeDh (IN OU
**DhContext)
EFI_STATUS IpSecCryptoIoGenerateRando
UINT8 *OutBuffer, IN UINT
VOID IpSecCryptoIoAuthDataWith
UINT8 *InData, IN UINTN I
UINT8 *PrivateKey, IN UINT
PrivateKeySize, IN UINT8 *
UINTN KeyPwdSize, OUT U
IN OUT UINTN *OutDataSiz
BOOLEAN IpSecCryptoIoVerifySignData
(IN UINT8 *InCert, IN UINT
UINT8 *InCa, IN UINTN Ca
*InData, IN UINTN InDataSi
*Singnature, IN UINTN SigS
EFI_STATUS IpSecCryptoIoGetPublicKeyF
UINT8 *InCert, IN UINTN C
UINT8 **PublicKey, OUT U
*PublicKeyLen)
EFI_STATUS IpSecCryptoIoGetSubjectFro
*InCert, IN UINTN CertSize,
**CertSubject, OUT UINTN

Detailed Description

Definitions related to the Cryptographic Operations in IPsec.

Copyright (c) 2009 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Authentication Algorithm Definition The number value definition is aligned to IANA assignment

Referenced by IpSecCryptoIoHash(), and IpSecCryptoIoHmac().

#define IKE_AALG_NULL 0xFB

Referenced by IpSecCryptoIoHash(), and IpSecCryptoIoHmac().

#define IKE_AALG_SHA1HMAC 0x02

Referenced by Ikev2GenerateCertificatePayload(), IpSecCryptoIoHash(), and IpSecCryptoIoHmac().

#define IKE_EALG_3DESCBC 0x03

Referenced by IpSecCryptoIoDecrypt(), and IpSecCryptoIoEncrypt().

#define IKE_EALG_AESCBC 0x0C

Referenced by IpSecCryptoIoDecrypt(), and IpSecCryptoIoEncrypt().

#define IKE_EALG_NONE 0x00
Encryption Algorithm Definition The number value definition is aligned to IANA assignment

Referenced by IpSecCryptoIoDecrypt(), and IpSecCryptoIoEncrypt().

#define IKE_EALG_NULL 0x0B

Referenced by IpSecCryptoIoDecrypt(), and IpSecCryptoIoEncrypt().

#define IPSEC_AUTH_ALGORITHM_LIST_SIZE 3

Referenced by IpSecGetHmacDigestLength(), IpSecGetIcvLength(), and IpSecGetIndexFromAuthList().

#define IPSEC_ENCRYPT_ALGORITHM_LIST_SIZE 4

Referenced by IpSecGetEncryptBlockSize(), IpSecGetEncryptIvLength(), IpSecGetEncryptKeyLength(), and IpSecGetIndexFromEncList().

#define IPSEC_HASH_ALGORITHM_LIST_SIZE 3

Typedef Documentation

typedef struct _AUTH_ALGORITHM AUTH_ALGORITHM

typedef BOOLEAN(EFIAPI * CRYPTO_CIPHER_DECRYPT)(IN VOID *Context, IN CONST UINT8 *InData, IN UINTN InputSize, IN CONST UINT8 *Ivec, OUT UINT8 *OutData)
Prototype of Cipher decryption. Decrypts cipher message with specified cipher.

If Context is NULL, then ASSERT(). if InData is NULL, then ASSERT(). If Size of input data is not a multiple of a certaion block size , then ASSERT().

Parameters:

[in]	*Context*	The specified Context.
[in]	*InData*	The input ciphertext data to be decrypted.
[in]	*InputSize*	The InData size.
[in]	*Ivec*	Pointer to the Initial Vector data for decryption.
[out]	*OutData*	The resultant decrypted plaintext.

Return values:

TRUE Decryption successful.

typedef BOOLEAN(EFIAPI * CRYPTO_CIPHER_ENCRYPT)(IN VOID *Context, IN CONST UINT8 *InData, IN UINTN InputSize, IN CONST UINT8 *Ivec, OUT UINT8 *OutData)

Prototype of Cipher encryption. Encrypts plaintext message with the specified cipher.

If Context is NULL, then ASSERT(). if InData is NULL, then ASSERT(). If Size of input data is not multiple of Cipher algorithm related block size, then ASSERT().

Parameters:

[in]	*Context*	The specified Context.
[in]	*InData*	The input plaintext data to be encrypted.
[in]	*InputSize*	The size of input data.
[in]	*Ivec*	Pointer to Initial Vector data for encryption.
[out]	*OutData*	The resultant encrypted ciphertext.

Return values:

TRUE Encryption successful.

typedef UINTN(EFIAPI * CRYPTO_CIPHER_GETCONTEXTSIZE)(VOID)

Prototype of Block Cipher GetContextSize.

Retrieves the size, in bytes, of the context buffer required.

Returns:

The size, in bytes, of the context buffer required.

typedef BOOLEAN(EFIAPI * CRYPTO_CIPHER_INIT)(IN OUT VOID *Context, IN CONST UINT8 *Key, IN UINTN KeyBits)

Prototype of Block Cipher initiation. Intializes the user-supplied key as the specifed context (key materials) for both encryption and decryption operations.

If Context is NULL, then ASSERT(). If Key is NULL, then generate random key for usage.

Parameters:

[in,out]	*Context*	The specified Context.
[in]	*Key*	User-supplied cipher key.
[in]	*KeyBits*	Key length in bits.

Return values:

TRUE Block Cipher Initialization was successful.

typedef BOOLEAN(EFIAPI * CRYPTO_HASH_FINAL)(IN OUT VOID *Context, OUT UINT8 *HashValue)

Prototype of Hash Finalization.

Completes computation of the digest value.

This function completes hash computation and retrieves the digest value into the specified memory. After this function has been called, the context cannot be used again. context should be already correctly intialized by HashInit(), and should not be finalized by HashFinal(). Behavior with invalid context is undefined.

If Context is NULL, then ASSERT(). If HashValue is NULL, then ASSERT().

Parameters:

`[in,out]`	*Context*	Pointer to the specified context.
`[out]`	*HashValue*	Pointer to a buffer that receives the digest value.

Return values:

TRUE	digest computation succeeded.
FALSE	digest computation failed.

typedef UINTN(EFIAPI * CRYPTO_HASH_GETCONTEXTSIZE)(VOID)

Prototype of Hash ContextSize.

Retrieves the size, in bytes, of the context buffer required for specified hash operations.

Returns:

The size, in bytes, of the context buffer required for certain hash operations.

typedef BOOLEAN(EFIAPI * CRYPTO_HASH_INIT)(OUT VOID *Context)

Prototype of Hash Initiate.

Initializes user-supplied memory pointed by Context as specified hash context for subsequent use.

If Context is NULL, then ASSERT().

Parameters:

`[out]`	*Context*	Pointer to specified context being initialized.

Return values:

TRUE	context initialization succeeded.
FALSE	context initialization failed.

typedef BOOLEAN(EFIAPI * CRYPTO_HASH_UPDATE)(IN OUT VOID *Context, IN CONST VOID *Data, IN UINTN DataSize)

Prototype of Hash Update

Digests the input data and updates hash context.

This function performs digest on a data buffer of the specified size. It can be called multiple times to compute the digest of long or discontinuous data streams. Context should be already correctly intialized by HashInit(),

and should not be finalized by HashFinal(). Behavior with invalid context is undefined.

If Context is NULL, then ASSERT().

Parameters:

[in,out]	*Context*	Pointer to the specified context.
[in]	*Data*	Pointer to the buffer containing the data to be hashed.
[in]	*DataSize*	Size of Data buffer in bytes.

Return values:

TRUE	data digest succeeded.
FALSE	data digest failed.

typedef BOOLEAN(EFIAPI * CRYPTO_HMAC_FINAL)(IN OUT VOID *Context, OUT UINT8 *HmacValue)

Prototype of HMAC finallization. Terminate a HMAC message digest operation and output the message digest.

If Context is NULL, then ASSERT(). If HashValue is NULL, then ASSERT().

Parameters:

[in,out]	*Context*	The specified Context.
[out]	*HmacValue*	Pointer to a 16-byte message digest output buffer.

Return values:

TRUE	Finalized successfully.

typedef UINTN(EFIAPI * CRYPTO_HMAC_GETCONTEXTSIZE)(VOID)

Prototype of HMAC GetContextSize.

Retrieves the size, in bytes, of the context buffer required.

Returns:

The size, in bytes, of the context buffer required.

typedef BOOLEAN(EFIAPI * CRYPTO_HMAC_INIT)(OUT VOID *Context, IN CONST UINT8 *Key, IN UINTN KeySize)

Prototype of HMAC Operation Initiating.

Initialization with a new context.

Parameters:

[out]	*Context*	Input Context.
[in]	*Key*	Pointer to the key for HMAC.
[in]	*KeySize*	The length of the Key in bytes.

Return values:

TRUE	Initialization Successfully.

typedef BOOLEAN(EFIAPI * CRYPTO_HMAC_UPDATE)(IN OUT VOID *Context, IN CONST VOID *Data, IN UINTN DataLength)

Prototype of HMAC update. HMAC update operation. Continue an HMAC message digest operation, processing another message block, and updating the HMAC context.

If Context is NULL, then ASSERT(). If Data is NULL, then ASSERT().

Parameters:

[in,out]	*Context*	The Specified Context.
[in,out]	*Data*	The Input Data to be digested.
[in]	*DataLength*	The length, in bytes, of Data.

Return values:

TRUE	Update data successfully.
FALSE	The Context has been finalized.

typedef struct _ENCRYPT_ALGORITHM ENCRYPT_ALGORITHM

typedef struct _HASH_ALGORITHM HASH_ALGORITHM

Function Documentation

```
VOID IpSecCryptoIoAuthDataWithCertificate ( IN UINT8 *          InData,
                                            IN UINTN            InDataSize,
                                            IN UINT8 *          PrivateKey,
                                            IN UINTN            PrivateKeySize,
                                            IN UINT8 *          KeyPassWord,
                                            IN UINTN            KeyPwdSize,
                                            OUT UINT8 **        OutData,
                                            IN OUT UINTN *      OutDataSize
                                          )
```

Authenticate data with the certificate.

Parameters:

[in]	*InData*	Pointer to the Data to be signed.
[in]	*InDataSize*	InData size in bytes.
[in]	*PrivateKey*	Pointer to the private key.
[in]	*PrivateKeySize*	The size of Private Key in bytes.
[in]	*KeyPassWord*	Pointer to the password for retrieving private key.
[in]	*KeyPwdSize*	The size of Key Password in bytes.
[out]	*OutData*	The pointer to the signed data.
[in,out]	*OutDataSize*	Pointer to contain the size of out data.

Referenced by Ikev2CertGenerateAuthPayload().

```
EFI_STATUS IpSecCryptoIoDecrypt     (                   AlgorithmId,
```

```
                                    IN CONST UINT8
                                    IN CONST UINT8 *    Key,
                                    IN CONST UINTN      KeyBits,
                                    IN CONST UINT8 *    Ivec,
                                    IN UINT8 *          InData,
                                    IN UINTN            InDataLength,
                                    OUT UINT8 *         OutData
                                )
```

Decrypts the buffer.

This function calls relevant Decryption interface from CryptoLib according to the input alogrithm ID. The InData should be multiple of block size. This function doesn't perform the padding. If it has the Ivec data, the length of it should be same with the block size. The block size is different from the different algorithm.

Parameters:

[in]	*AlgorithmId*	The Alogrithem identification defined in RFC.
[in]	*Key*	Pointer to the buffer containing encrypting key.
[in]	*KeyBits*	The length of the key in bits.
[in]	*Ivec*	Point to the buffer containning the Initializeion Vector (IV) data.
[in]	*InData*	Point to the buffer containing the data to be decrypted.
[in]	*InDataLength*	The length of InData in Bytes.
[out]	*OutData*	Pointer to the buffer that receives the decryption output.

Return values:

EFI_UNSUPPORTED	The input Algorithm is not supported.
EFI_OUT_OF_RESOURCE	The required resource can't be allocated.
EFI_SUCCESS	The operation completed successfully.

References _ENCRYPT_ALGORITHM::CipherGetContextSize, IKE_EALG_3DESCBC, IKE_EALG_AESCBC, IKE_EALG_NONE, IKE_EALG_NULL, IpSecGetIndexFromEncList(), and mIpsecEncryptAlgorithmList.

Referenced by Ikev2DecryptPacket(), and IpSecEspInboundPacket().

```
EFI_STATUS IpSecCryptoIoDhComputeKey    (   IN OUT UINT8 *      DhContext,
                                            IN CONST UINT8 *    PeerPublicKey,
                                            IN UINTN            PeerPublicKeySize,
                                            OUT UINT8 *         Key,
                                            IN OUT UINTN *      KeySize
                                        )
```

Generates exchanged common key.

Given peer's public key, this function computes the exchanged common key, based on its own context including value of prime modulus and random secret exponent.

Parameters:

[in,out]	*DhContext*	Pointer to the DH context.
[in]	*PeerPublicKey*	Pointer to the peer's Public Key.
[in]	*PeerPublicKeySize*	Size of peer's public key in bytes.
[out]	*Key*	Pointer to the buffer to receive generated key.
[in,out]	*KeySize*	For in, the size of Key buffer in bytes. For out, the size of data returned in Key buffer in bytes.

Return values:

EFI_SUCCESS	The operation perfoms successfully.
Otherwise	The operation is failed.

Referenced by Ikev2GenerateSaDhComputeKey().

```
EFI_STATUS IpSecCryptoIoDhGetPublicKey ( IN OUT UINT8 **      DhContext,
                                         IN UINTN             Generator,
                                         IN UINTN             PrimeLength,
                                         IN CONST UINT8 *     Prime,
                                         OUT UINT8 *          PublicKey,
                                         IN OUT UINTN *       PublicKeySize
                                       )
```

Generates the Diffie-Hellman public key.

This function first initiate a DHContext, then call the DhSetParameter() to set the prime and primelenght, at end call the DhGenerateKey() to generates random secret exponent, and computes the public key. The output returned via parameter PublicKey and PublicKeySize. DH context is updated accordingly. If the PublicKey buffer is too small to hold the public key, EFI_INVALID_PARAMETER is returned and PublicKeySize is set to the required buffer size to obtain the public key.

Parameters:

[in,out]	*DhContext*	Pointer to the DH context.
[in]	*Generator*	Vlaue of generator.
[in]	*PrimeLength*	Length in bits of prime to be generated.
[in]	*Prime*	Pointer to the buffer to receive the generated prime number.
[out]	*PublicKey*	Pointer to the buffer to receive generated public key.
[in,out]	*PublicKeySize*	For in, the size of PublicKey buffer in bytes. For out, the size of data returned in PublicKey buffer in bytes.

Return values:

EFI_SUCCESS	The operation perfoms successfully.
Otherwise	The operation is failed.

Referenced by Ikev2GenerateSaDhPublicKey().

```
EFI_STATUS IpSecCryptoIoEncrypt ( IN CONST UINT8      AlgorithmId,
```

```
                                    IN CONST UINT8 *   Key,
                                    IN CONST UINTN     KeyBits,
                                    IN CONST UINT8 *   Ivec,
                                    IN UINT8 *         InData,
                                    IN UINTN           InDataLength,
                                    OUT UINT8 *        OutData
                                  )
```

Encrypt the buffer.

This function calls relevant encryption interface from CryptoLib according to the input alogrithm ID. The InData should be multiple of block size. This function doesn't perform the padding. If it has the Ivec data, the length of it should be same with the block size. The block size is different from the different algorithm.

Parameters:

[in]	*AlgorithmId*	The Alogrithem identification defined in RFC.
[in]	*Key*	Pointer to the buffer containing encrypting key.
[in]	*KeyBits*	The length of the key in bits.
[in]	*Ivec*	Point to the buffer containning the Initializeion Vector (IV) data.
[in]	*InData*	Point to the buffer containing the data to be encrypted.
[in]	*InDataLength*	The length of InData in Bytes.
[out]	*OutData*	Point to the buffer that receives the encryption output.

Return values:

EFI_UNSUPPORTED	The input Algorithm is not supported.
EFI_OUT_OF_RESOURCE	The required resource can't be allocated.
EFI_SUCCESS	The operation completed successfully.

References _ENCRYPT_ALGORITHM::CipherGetContextSize, IKE_EALG_3DESCBC, IKE_EALG_AESCBC, IKE_EALG_NONE, IKE_EALG_NULL, IpSecGetIndexFromEncList(), and mIpsecEncryptAlgorithmList.

Referenced by Ikev2EncryptPacket(), and IpSecEspOutboundPacket().

```
EFI_STATUS IpSecCryptoIoFreeDh ( IN OUT UINT8 **   DhContext   )
```

Releases the DH context. If DhContext is NULL, return EFI_INVALID_PARAMETER.

Parameters:

[in,out]	*DhContext*	Pointer to the DH context to be freed.

Return values:

EFI_SUCCESS	The operation perfoms successfully.
EFI_INVALID_PARAMETER	The DhContext is NULL.

Referenced by Ikev2DhBufferFree().

```
EFI_STATUS IpSecCryptoIoGenerateRandomBytes ( OUT UINT8 *   OutBuffer,
                                              IN UINTN      Bytes
                                            )
```

Generates random numbers of specified size.

If the Random Generator wasn't initiated, initiate it first, then call RandomBytes.

Parameters:

[out]	*OutBuffer*	Pointer to buffer to receive random value.
[in]	*Bytes*	Size of randome bytes to generate.

Return values:

EFI_SUCCESS	The operation perfoms successfully.
Otherwise	The operation is failed.

References mInitialRandomSeed.

Referenced by IkeGenerateCookie(), IkeGenerateIv(), IkeGenerateNonce(), and IpSecGenerateIv().

```
EFI_STATUS IpSecCryptoIoGetPublicKeyFromCert ( IN UINT8 *      InCert,
                                               IN UINTN        CertLen,
                                               OUT UINT8 **    PublicKey,
                                               OUT UINTN *     PublicKeyLen
                                             )
```

Retrieves the RSA Public Key from one X509 certificate (DER format only).

Parameters:

[in]	*InCert*	Pointer to the certificate.
[in]	*CertLen*	The size of the certificate in bytes.
[out]	*PublicKey*	Pointer to the retrieved public key.
[out]	*PublicKeyLen*	Size of Public Key in bytes.

Return values:

EFI_SUCCESS	Successfully get the public Key.
EFI_INVALID_PARAMETER	The CA certificate is malformed.

Retrieves the RSA Public Key from one X509 certificate (DER format only).

Parameters:

[in]	*InCert*	Pointer to the certificate.
[in]	*CertLen*	The size of the certificate in bytes.
[out]	*PublicKey*	Pointer to the retrieved public key.
[out]	*PublicKeyLen*	Size of Public Key in bytes.

Return values:

EFI_SUCCESS	Successfully get the public Key.
EFI_INVALID_PARAMETER	The certificate is malformed.

Referenced by Ikev2GenerateCertificatePayload().

```
EFI_STATUS IpSecCryptoIoGetSubjectFromCert ( IN UINT8 *      InCert,
                                             IN UINTN        CertSize,
                                             OUT UINT8 **    CertSubject,
                                             OUT UINTN *     SubjectSize
```

)

Retrieves the subject name from one X509 certificate (DER format only).

Parameters:

[in]	*InCert*	Pointer to the X509 certificate.
[in]	*CertSize*	The size of the X509 certificate in bytes.
[out]	*CertSubject*	Pointer to the retrieved certificate subject.
[out]	*SubjectSize*	The size of Certificate Subject in bytes.

Return values:

EFI_SUCCESS	Retrieved the certificate subject successfully.
EFI_INVALID_PARAMETER	The certificate is malformed.

Referenced by Ikev2GenerateCertIdPayload().

```
EFI_STATUS IpSecCryptoIoHash ( IN CONST UINT8                AlgorithmId,
                               IN HASH_DATA_FRAGMENT *       InDataFragment,
                               IN UINTN                      FragmentCount,
                               OUT UINT8 *                   OutData,
                               IN UINTN                      OutDataSize
                             )
```

Digests the Payload and store the result into the OutData.

This function calls relevant Hash interface from CryptoLib according to the input alogrithm ID. It computes all datas from InDataFragment and output the result into the OutData buffer. If the OutDataSize is larger than the related Hash alogrithm output size, return EFI_INVALID_PARAMETER.

Parameters:

[in]	*AlgorithmId*	The authentication Identification.
[in]	*InDataFragment*	A list contains all data to be authenticated.
[in]	*FragmentCount*	The size of the InDataFragment.
[out]	*OutData*	For in, the buffer to receive the output data. For out, the buffer contains the authenticated data.
[in]	*OutDataSize*	The size of the buffer of OutData.

Return values:

EFI_UNSUPPORTED	If the AuthAlg is not in the support list.
EFI_SUCCESS	Authenticated the payload successfully.
EFI_INVALID_PARAMETER	If the OutDataSize is larger than the related Hash algorithm could handle.
otherwise	Authentication of the payload failed.

References _HASH_ALGORITHM::HashGetContextSize, IKE_AALG_NONE, IKE_AALG_NULL, IKE_AALG_SHA1HMAC, IpSecGetHmacDigestLength(), IpSecGetIndexFromAuthList(), and mIpsecHashAlgorithmList.

Referenced by Ikev2CertGenerateAuthPayload(), and Ikev2GenerateCertificatePayload().

```
EFI_STATUS IpSecCryptoIoHmac ( IN CONST UINT8                AlgorithmId,
```

```
                                IN CONST UINT8 *              Key,
                                IN UINTN                      KeyLength,
                                IN HASH_DATA_FRAGMENT *       InDataFragment,
                                IN UINTN                      FragmentCount,
                                OUT UINT8 *                   OutData,
                                IN UINTN                      OutDataSize
                              )
```

Digests the Payload with key and store the result into the OutData.

This function calls relevant Hmac interface from CryptoLib according to the input alogrithm ID. It computes all datas from InDataFragment and output the result into the OutData buffer. If the OutDataSize is larger than the related HMAC alogrithm output size, return EFI_INVALID_PARAMETER.

Parameters:

[in]	*AlgorithmId*	The authentication Identification.
[in]	*Key*	Pointer of the authentication key.
[in]	*KeyLength*	The length of the Key in bytes.
[in]	*InDataFragment*	The list contains all data to be authenticated.
[in]	*FragmentCount*	The size of the InDataFragment.
[out]	*OutData*	For in, the buffer to receive the output data. For out, the buffer contains the authenticated data.
[in]	*OutDataSize*	The size of the buffer of OutData.

Return values:

EFI_UNSUPPORTED	If the AuthAlg is not in the support list.
EFI_INVALID_PARAMETER	The OutData buffer size is larger than algorithm digest size.
EFI_SUCCESS	Authenticate the payload successfully.
otherwise	Authentication of the payload fails.

References _AUTH_ALGORITHM::HmacGetContextSize, IKE_AALG_NONE, IKE_AALG_NULL, IKE_AALG_SHA1HMAC, IpSecGetHmacDigestLength(), IpSecGetIndexFromAuthList(), and mIpsecAuthAlgorithmList.

Referenced by Ikev2CertGenerateAuthPayload(), Ikev2DecryptPacket(), Ikev2EncryptPacket(), Ikev2GenerateSaKeys(), Ikev2PskGenerateAuthPayload(), Ikev2SaGenerateKey(), IpSecEspAuthVerifyPayload(), and IpSecEspOutboundPacket().

```
BOOLEAN IpSecCryptoIoVerifySignDataByCertificate ( IN UINT8 *   InCert,
                                                   IN UINTN     CertLen,
                                                   IN UINT8 *   InCa,
                                                   IN UINTN     CaLen,
                                                   IN UINT8 *   InData,
                                                   IN UINTN     InDataSize,
                                                   IN UINT8 *   Singnature,
                                                   IN UINTN     SigSize
                                                 )
```

Verify the singed data with the public key which is contained in a certificate.

Parameters:

[in]	*InCert*	Pointer to the Certificate which contains the public key.
[in]	*CertLen*	The size of Certificate in bytes.
[in]	*InCa*	Pointer to the CA certificate
[in]	*CaLen*	The size of CA certificate in bytes.
[in]	*InData*	Pointer to octect message hash to be checked.
[in]	*InDataSize*	Size of the message hash in bytes.
[in]	*Singnature*	The pointer to the RSA PKCS1-V1_5 signature to be verifed.
[in]	*SigSize*	Size of signature in bytes.

Return values:

TRUE	Valid signature encoded in PKCS1-v1_5.
FALSE	Invalid signature or invalid RSA context.

Referenced by Ikev2AuthCertParser().

```
EFI_STATUS IpSecGenerateIv ( IN UINT8 *  IvBuffer,
                             IN UINTN    IvSize
                           )
```

Generate a random data for IV. If the IvSize is zero, not needed to create IV and return EFI_SUCCESS.

Parameters:

[in]	*IvBuffer*	The pointer of the IV buffer.
[in]	*IvSize*	The IV size in bytes.

Return values:

EFI_SUCCESS	Create random data for IV.

Generate a random data for IV. If the IvSize is zero, not needed to create IV and return EFI_SUCCESS.

Parameters:

[in]	*IvBuffer*	The pointer of the IV buffer.
[in]	*IvSize*	The IV size in bytes.

Return values:

EFI_SUCCESS	Create a random data for IV.

References IpSecCryptoIoGenerateRandomBytes().

Referenced by IpSecEspOutboundPacket().

```
UINTN IpSecGetEncryptBlockSize ( IN UINT8  AlgorithmId )
```

Get the block size of specified encryption alogrithm.

Parameters:

[in]	*AlgorithmId*	The encryption algorithm ID.

Returns:

The value of block size.

References _ENCRYPT_ALGORITHM::BlockSize, IPSEC_ENCRYPT_ALGORITHM_LIST_SIZE, and mIpsecEncryptAlgorithmList.

Referenced by Ikev2DecryptPacket(), Ikev2EncryptPacket(), IpSecEspInboundPacket(), and IpSecEspOutboundPacket().

UINTN IpSecGetEncryptIvLength (IN UINT8 *AlgorithmId*)

Get the IV size of specified encryption alogrithm.

Parameters:

[in] *AlgorithmId* The encryption algorithm ID.

Returns:

The value of IV size.

Get the IV size of the specified encryption alogrithm.

Parameters:

[in] *AlgorithmId* The encryption algorithm ID.

Returns:

The value of IV size.

References IPSEC_ENCRYPT_ALGORITHM_LIST_SIZE, _ENCRYPT_ALGORITHM::IvLength, and mIpsecEncryptAlgorithmList.

Referenced by IpSecEspInboundPacket(), and IpSecEspOutboundPacket().

UINTN IpSecGetEncryptKeyLength (IN UINT8 *AlgorithmId*)

Get the required key length of the specified encryption alogrithm.

Parameters:

[in] *AlgorithmId* The encryption algorithm ID.

Returns:

The value of key length.

Get the key length of the specified encryption alogrithm.

Parameters:

[in] *AlgorithmId* The encryption algorithm ID.

Returns:

The value of key length.

References IPSEC_ENCRYPT_ALGORITHM_LIST_SIZE, _ENCRYPT_ALGORITHM::KeyLength, and mIpsecEncryptAlgorithmList.

Referenced by Ikev2GenerateChildSaKeys(), Ikev2GenerateSaKeys(), Ikev2InitializeSaData(), and Ikev2ParseProposalData().

UINTN IpSecGetHmacDigestLength (IN UINT8 *AlgorithmId*)
Get the HMAC digest length by the specified Algorithm ID.

Parameters:
`[in]` *AlgorithmId* The specified Alogrithm ID.

Returns:
The digest length of the specified Authentication Algorithm ID.

References _AUTH_ALGORITHM::DigestLength, IPSEC_AUTH_ALGORITHM_LIST_SIZE, and mIpsecAuthAlgorithmList.

Referenced by Ikev2CertGenerateAuthPayload(), Ikev2GenerateCertificatePayload(), Ikev2GenerateChildSaKeys(), Ikev2GenerateSaKeys(), Ikev2PskGenerateAuthPayload(), Ikev2SaGenerateKey(), IpSecCryptoIoHash(), and IpSecCryptoIoHmac().

UINTN IpSecGetIcvLength (IN UINT8 *AlgorithmId*)
Get the ICV size of the specified Authenticaion alogrithm.

Parameters:
`[in]` *AlgorithmId* The Authentication algorithm ID.

Returns:
The value of ICV size.

References _AUTH_ALGORITHM::IcvLength, IPSEC_AUTH_ALGORITHM_LIST_SIZE, and mIpsecAuthAlgorithmList.

Referenced by Ikev2DecryptPacket(), Ikev2EncryptPacket(), IpSecEspInboundPacket(), and IpSecEspOutboundPacket().

Generated on Mon Sep 28 08:49:55 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/IpSecDebug.c File Reference

Functions

VOID IpSecDumpAddress (IN UINTN Level, IN EFI_IP_ADDRESS *Ip, IN UINT8 IpVersion)
VOID IkeDumpState (IN UINT32 Previous, IN UINT32 Current, IN UINT8 IkeVersion)
VOID IpSecDumpPacket (IN IKE_PACKET *Packet, IN EFI_IPSEC_TRAFFIC_DIR Direction, IN UINT8 IpVersion)
VOID IpSecDumpPayload (IN IKE_PAYLOAD *IkePayload, IN UINT8 IkeVersion)
VOID IpSecDumpBuf (IN CHAR8 *Title, IN UINT8 *Data, IN UINTN DataSize)

Variables

CHAR8 * mIkev1StateStr [IKE_STATE_NUM]
CHAR8 * mIkev2StateStr [IKE_STATE_NUM]
CHAR8 * mExchangeStr []
CHAR8 * mIkev1PayloadStr []
CHAR8 * mIkev2PayloadStr []

Detailed Description

The Interfaces of IPsec debug information printing.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
VOID IkeDumpState ( IN UINT32  Previous,
                    IN UINT32  Current,
                    IN UINT8   IkeVersion
                  )
```

Print IKE Current states.

Parameters:

[in]	*Previous*	The Previous state of IKE.
[in]	*Current*	The current state of IKE.
[in]	*IkeVersion*	The version of IKE.

References IKE_STATE_NUM, mIkev1StateStr, and mIkev2StateStr.

```
VOID IpSecDumpAddress ( IN UINTN                Level,
                        IN EFI_IP_ADDRESS *     Ip,
                        IN UINT8                IpVersion
                      )
```

Print the IP address.

Parameters:

[in]	*Level*	Debug print error level. Pass to DEBUG().
[in]	*Ip*	Point to a specified IP address.
[in]	*IpVersion*	The IP Version.

Referenced by IpSecDumpPacket().

```
VOID IpSecDumpBuf ( IN CHAR8 *    Title,
                    IN UINT8 *    Data,
                    IN UINTN      DataSize
                  )
```

Print the buffer in form of Hex.

Parameters:

[in]	*Title*	The strings to be printed before the data of the buffer.
[in]	*Data*	Points to buffer to be printed.
[in]	*DataSize*	The size of the buffer to be printed.

References IPSEC_DEBUG_BYTE_PER_LINE.

Referenced by Ikev2CertGenerateAuthPayload(), and IpSecDumpPayload().

```
VOID IpSecDumpPacket ( IN IKE_PACKET *              Packet,
                       IN EFI_IPSEC_TRAFFIC_DIR     Direction,
                       IN UINT8                     IpVersion
                     )
```

Print the IKE Packet.

Parameters:

[in]	*Packet*	Point to IKE packet to be printed.
[in]	*Direction*	Point to the IKE packet is inbound or outbound.
[in]	*IpVersion*	Specified IP Version.

References IKE_XCG_TYPE_AUTH, IKE_XCG_TYPE_CREATE_CHILD_SA, IKE_XCG_TYPE_IDENTITY_PROTECT, IKE_XCG_TYPE_INFO, IKE_XCG_TYPE_INFO2, IKE_XCG_TYPE_QM, IKE_XCG_TYPE_SA_INIT, IpSecDumpAddress(), and mExchangeStr.

VOID IpSecDumpPayload (IN IKE_PAYLOAD * *IkePayload*,
IN UINT8 *IkeVersion*
)

Print the IKE Paylolad.

Parameters:

[in] *IkePayload* Point to payload to be printed.
[in] *IkeVersion* The specified version of IKE.

References IpSecDumpBuf(), mIkev1PayloadStr, and mIkev2PayloadStr.

Variable Documentation

CHAR8* mExchangeStr[]

Initial value:

```
{
 "IKEv1 Main Exchange",
 "IKEv1 Info Exchange",
 "IKEv1 Quick Exchange",
 "IKEv2 Initial Exchange",
 "IKEv2 Auth Exchange",
 "IKEv2 Create Child Exchange",
 "IKEv2 Info Exchange",
 "IKE   Unknow Exchange"
}
```

Referenced by IpSecDumpPacket().

CHAR8* mIkev1PayloadStr[]

Initial value:

```
{
 "IKEv1 None Payload",
 "IKEv1 SA Payload",
 "IKEv1 Proposal Payload",
 "IKEv1 Transform Payload",
 "IKEv1 KE Payload",
 "IKEv1 ID Payload",
 "IKEv1 Certificate Payload",
 "IKEv1 Certificate Request Payload",
 "IKEv1 Hash Payload",
 "IKEv1 Signature Payload",
 "IKEv1 Nonce Payload",
 "IKEv1 Notify Payload",
 "IKEv1 Delete Payload",
 "IKEv1 Vendor Payload"
}
```

Referenced by IpSecDumpPayload().

CHAR8* mIkev1StateStr[IKE_STATE_NUM]

Initial value:

```
{
 "IKEv1_MAIN_1",
 "IKEv1_MAIN_2",
 "IKEv1_MAIN_3",
```

```
  "IKEv1_MAIN_ESTABLISHED",
  "IKEv1_QUICK_1",
  "IKEv1_QUICK_2",
  "IKEv1_QUICK_ESTABLISHED"
}
```

Referenced by IkeDumpState().

CHAR8* mIkev2PayloadStr[]

Initial value:

```
 {
  "IKEv2 SA Payload",
  "IKEv2 Key Payload",
  "IKEv2 Identity Initial Payload",
  "IKEv2 Identity Respond Payload",
  "IKEv2 Certificate Payload",
  "IKEv2 Certificate Request Payload",
  "IKEv2 Auth Payload",
  "IKEv2 Nonce Payload",
  "IKEv2 Notify Payload",
  "IKEv2 Delet Payload",
  "IKEv2 Vendor Payload",
  "IKEv2 Traffic Selector Initiator Payload",
  "IKEv2 Traffic Selector Respond Payload",
  "IKEv2 Encrypt Payload",
  "IKEv2 Configuration Payload",
  "IKEv2 Extensible Authentication Payload"
}
```

Referenced by IpSecDumpPayload().

CHAR8* mIkev2StateStr[IKE_STATE_NUM]

Initial value:

```
 {
  "IKEv2_STATE_INIT",
  "IKEv2_STATE_AUTH",
  "IKEv2_STATE_SA_ESTABLISH",
  "IKEv2_STATE_CREATE_CHILD",
  "IKEv2_STATE_SA_REKEYING",
  "IKEv2_STATE_CHILD_SA_ESTABLISHED",
  "IKEv2_STATE_SA_DELETING"
}
```

Referenced by IkeDumpState().

NetworkPkg/IpSecDxe/IpSecDebug.h File Reference

Defines

"memItemRight" valign="bottom">IPSEC_DUMP_ADDRESS(Level, Ip, Version) IpSecDumpAddress (Level, Ip,

Functions

Detailed Description

The definition of functions and MACROs used for IPsec debug information printting.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by IkeDumpState().

```
#define IKEV1_DUMP_STATE ( Previous,
                           Current ) IkeDumpState (Previous, Current, 1)
```

```
#define IKEV2_DUMP_PAYLOAD ( IkePayload ) IpSecDumpPayload (IkePayload, 2)
```

Referenced by Ikev2DecodePayload(), and Ikev2EncodePayload().

```
#define IKEV2_DUMP_STATE ( Previous,
                           Current ) IkeDumpState (Previous, Current, 2)
```

Referenced by Ikev2AuthCertParser(), Ikev2AuthPskParser(), Ikev2ChildSaSessionCreate(), Ikev2HandleSa(), Ikev2InitPskParser(), Ikev2LifetimeNotify(), Ikev2NegotiateChildSa(), and Ikev2NegotiateSa().

```
#define IPSEC_DEBUG_BYTE_PER_LINE 8
```

Referenced by IpSecDumpBuf().

```
#define IPSEC_DUMP_ADDRESS ( Level,
                             Ip,
                             Version ) IpSecDumpAddress (Level, Ip, Version)
```

```
#define IPSEC_DUMP_BUF ( Title,
                         Data,
                         DataSize ) IpSecDumpBuf (Title, Data, DataSize)
```

Referenced by Ikev2DecodePacket(), Ikev2DecryptPacket(), Ikev2EncryptPacket(), Ikev2GenerateChildSaKeys(), Ikev2GenerateSaDhComputeKey(), Ikev2GenerateSaDhPublicKey(), and Ikev2GenerateSaKeys().

```
#define IPSEC_DUMP_PACKET ( Packet,
                            Direction,
                            IpVersion ) IpSecDumpPacket (Packet, Direction, IpVersion)
```

Referenced by Ikev2SendIkePacket().

```
#define IPSEC_DUMP_PAYLOAD ( IkePayload ) IpSecDumpPayload (IkePayload, 1)
```

Function Documentation

```
VOID IkeDumpState ( IN UINT32 Previous,
                    IN UINT32 Current,
                    IN UINT8  IkeVersion
                  )
```

Print IKE Current states.

Parameters:

[in]	*Previous*	The Previous state of IKE.
[in]	*Current*	The current state of IKE.
[in]	*IkeVersion*	The version of IKE.

References IKE_STATE_NUM, mIkev1StateStr, and mIkev2StateStr.

```
VOID IpSecDumpAddress ( IN UINTN            Level,
                        IN EFI_IP_ADDRESS * Ip,
                        IN UINT8            IpVersion
                      )
```

Print the IP address.

Parameters:

[in]	*Level*	Debug print error level. Pass to DEBUG().
[in]	*Ip*	Point to specified IP address.
[in]	*IpVersion*	The IP Version.

Print the IP address.

Parameters:

[in]	*Level*	Debug print error level. Pass to DEBUG().
[in]	*Ip*	Point to a specified IP address.
[in]	*IpVersion*	The IP Version.

Referenced by IpSecDumpPacket().

```
                    IN
VOID IpSecDumpBuf ( CHAR8  Title,
                    *
```

```
                    IN
                    UINT8 *   Data,
                    IN
                    UINTN     DataSize
                )
```

Print the buffer in form of Hex.

Parameters:

[in]	*Title*	The strings to be printed before the data of the buffer.
[in]	*Data*	Point to buffer to be printed.
[in]	*DataSize*	The size of the buffer to be printed.

Print the buffer in form of Hex.

Parameters:

[in]	*Title*	The strings to be printed before the data of the buffer.
[in]	*Data*	Points to buffer to be printed.
[in]	*DataSize*	The size of the buffer to be printed.

References IPSEC_DEBUG_BYTE_PER_LINE.

Referenced by Ikev2CertGenerateAuthPayload(), and IpSecDumpPayload().

```
VOID IpSecDumpPacket ( IN IKE_PACKET *              Packet,
                       IN EFI_IPSEC_TRAFFIC_DIR     Direction,
                       IN UINT8                     IpVersion
                     )
```

Print the IKE Packet.

Parameters:

[in]	*Packet*	Point to IKE packet to be printed.
[in]	*Direction*	Point to the IKE packet is inbound or outbound.
[in]	*IpVersion*	Specified IP Version.

References IKE_XCG_TYPE_AUTH, IKE_XCG_TYPE_CREATE_CHILD_SA, IKE_XCG_TYPE_IDENTITY_PROTECT, IKE_XCG_TYPE_INFO, IKE_XCG_TYPE_INFO2, IKE_XCG_TYPE_QM, IKE_XCG_TYPE_SA_INIT, IpSecDumpAddress(), and mExchangeStr.

```
VOID IpSecDumpPayload ( IN IKE_PAYLOAD *   IkePayload,
                        IN UINT8           IkeVersion
                      )
```

Print the IKE Paylolad.

Parameters:

[in]	*IkePayload*	Point to payload to be printed.
[in]	*IkeVersion*	The specified version of IKE.

References IpSecDumpBuf(), mIkev1PayloadStr, and mIkev2PayloadStr.

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/IpSecDriver.c File Reference

Functions

EFI_STATUS EFIAPI IpSecSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath, IN UINT8 IpVersion)

EFI_STATUS EFIAPI IpSecStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath, IN UINT8 IpVersion)

EFI_STATUS EFIAPI IpSecStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer, IN UINT8 IpVersion)

EFI_STATUS EFIAPI IpSec4DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI IpSec4DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI IpSec4DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)

EFI_STATUS EFIAPI IpSec6DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI IpSec6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI IpSec6DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)

VOID EFIAPI IpSecCleanupAllSa (IN EFI_EVENT Event, IN VOID *Context)

EFI_STATUS EFIAPI IpSecDriverEntryPoint (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)

Variables

EFI_DRIVER_BINDING_PROTOCOL gIpSec4DriverBinding

EFI_DRIVER_BINDING_PROTOCOL gIpSec6DriverBinding

Detailed Description

Driver Binding Protocol for IPsec Driver.

Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI
IpSec4DriverBindingStart (
    IN EFI_DRIVER_BINDING_PROTOCOL *   This,
    IN EFI_HANDLE                      ControllerHandle,
    IN EFI_DEVICE_PATH_PROTOCOL *      RemainingDevicePath
)
```

Start this driver on ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to bind driver to.
[in]	*RemainingDevicePath*	Optional parameter used to pick a specific child device to start.

Return values:

EFI_SUCCES	This driver is added to ControllerHandle
EFI_ALREADY_STARTED	This driver is already running on ControllerHandle
EFI_DEVICE_ERROR	The device could not be started due to a device error. Currently not implemented.
other	This driver does not support this device

References IpSecStart().

```
EFI_STATUS EFIAPI
IpSec4DriverBindingStop (
    IN EFI_DRIVER_BINDING_PROTOCOL *   This,
    IN EFI_HANDLE                      ControllerHandle,
    IN UINTN                           NumberOfChildren,
    IN EFI_HANDLE *                    ChildHandleBuffer
)
```

Stop this driver on ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of a device to stop the driver on.
[in]	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If the number of children is zero, stop the entire bus driver.

[in] *ChildHandleBuffer* List of Child Handles to Stop.

Return values:

EFI_SUCCES This driver removed ControllerHandle.
other This driver was not removed from this device.

References IpSecStop().

```
EFI_STATUS EFIAPI
IpSec4DriverBindingSupported
(
  IN EFI_DRIVER_BINDING_PROTOCOL *   This,
  IN EFI_HANDLE                      ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL *      RemainingDevicePath
)
```

Test to see if this driver supports ControllerHandle.

Parameters:

[in] *This* Protocol instance pointer.
[in] *ControllerHandle* Handle of device to test.
[in] *RemainingDevicePath* Optional parameter used to pick a specific child device to start.

Return values:

EFI_SUCCES This driver supports this device.
EFI_ALREADY_STARTED This driver is already running on this device.
other This driver does not support this device.

References IpSecSupported().

```
EFI_STATUS EFIAPI
IpSec6DriverBindingStart
(
  IN EFI_DRIVER_BINDING_PROTOCOL *   This,
  IN EFI_HANDLE                      ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL *      RemainingDevicePath
)
```

Start this driver on ControllerHandle.

Parameters:

[in] *This* Protocol instance pointer.
[in] *ControllerHandle* Handle of device to bind driver to.
[in] *RemainingDevicePath* Optional parameter used to pick a specific child device to start.

Return values:

EFI_SUCCES This driver is added to ControllerHandle
EFI_ALREADY_STARTED This driver is already running on ControllerHandle
EFI_DEVICE_ERROR The device could not be started due to a device error. Currently not implemented.
other This driver does not support this device

References IpSecStart().

```
EFI_STATUS EFIAPI
IpSec6DriverBindingStop (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     ControllerHandle,
  IN UINTN                          NumberOfChildren,
  IN EFI_HANDLE *                   ChildHandleBuffer
)
```

Stop this driver on ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of a device to stop the driver on.
[in]	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If the number of children is zero, stop the entire bus driver.
[in]	*ChildHandleBuffer*	List of Child Handles to Stop.

Return values:

EFI_SUCCES	This driver removed ControllerHandle.
other	This driver was not removed from this device.

References IpSecStop().

```
EFI_STATUS EFIAPI
IpSec6DriverBindingSupported (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
)
```

Test to see if this driver supports ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to test.
[in]	*RemainingDevicePath*	Optional parameter used to pick a specific child device to start.

Return values:

EFI_SUCCES	This driver supports this device.
EFI_ALREADY_STARTED	This driver is already running on this device.
other	This driver does not support this device.

References IpSecSupported().

```
VOID EFIAPI IpSecCleanupAllSa (
  IN EFI_EVENT   Event,
  IN VOID *      Context
)
```

This is a callback function when the mIpSecInstance.DisabledEvent is signaled.

Parameters:

[in]	*Event*	Event whose notification function is being invoked.

[in] *Context* Pointer to the notification function's context.

References IkeDeleteAllSas(), and _IPSEC_PRIVATE_DATA::IsIPsecDisabling.

Referenced by IpSecDriverEntryPoint().

```
EFI_STATUS EFIAPI IpSecDriverEntryPoint ( IN EFI_HANDLE          ImageHandle,
                                          IN EFI_SYSTEM_TABLE *  SystemTable
                                        )
```

This is the declaration of an EFI image entry point. This entry point is the same for UEFI Applications, UEFI OS Loaders, and UEFI Drivers, including both device drivers and bus drivers.

The entry point for IPsec driver which installs the driver binding, component name protocol, IPsec Config protcolon, and IPsec protocol in its ImageHandle.

Parameters:

[in] *ImageHandle* The firmware allocated handle for the UEFI image.
[in] *SystemTable* A pointer to the EFI System Table.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_ALREADY_STARTED	The IPsec driver has been already loaded.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
Others	The operation is failed.

References gIpSec4DriverBinding, gIpSec6DriverBinding, gIpSecComponentName, gIpSecComponentName2, _IPSEC_PRIVATE_DATA::Handle, _IPSEC_PRIVATE_DATA::Ikev1EstablishedList, _IPSEC_PRIVATE_DATA::Ikev1SessionList, _IPSEC_PRIVATE_DATA::Ikev2EstablishedList, _IPSEC_PRIVATE_DATA::Ikev2SessionList, _IPSEC_PRIVATE_DATA::ImageHandle, _IPSEC_PRIVATE_DATA::IpSec, IPSEC_PRIVATE_DATA_SIGNATURE, IpSecCleanupAllSa(), _IPSEC_PRIVATE_DATA::IpSecConfig, IpSecConfigInitialize(), mDpc, mIpSecInstance, _IPSEC_PRIVATE_DATA::Signature, _IPSEC_PRIVATE_DATA::Udp4List, and _IPSEC_PRIVATE_DATA::Udp6List.

```
EFI_STATUS EFIAPI IpSecStart ( IN EFI_DRIVER_BINDING_PROTOCOL *  This,
                               IN EFI_HANDLE                     ControllerHandle,
                               IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath,
                               IN UINT8                          IpVersion
                             )
```

Start this driver on ControllerHandle. This is the worker function for IpSec4(6)DriverbindingStart.

Parameters:

[in] *This*	Protocol instance pointer.
[in] *ControllerHandle*	Handle of device to bind driver to.
[in] *RemainingDevicePath*	Optional parameter used to pick a specific child device to start.
[in] *IpVersion*	IP_VERSION_4 or IP_VERSION_6.

Return values:

EFI_SUCCES	This driver is added to ControllerHandle
EFI_ALREADY_STARTED	This driver is already running on ControllerHandle

EFI_DEVICE_ERROR	The device could not be started due to a device error. Currently not implemented.
other	This driver does not support this device

References IkeOpenInputUdp4(), IkeOpenInputUdp6(), and IPSEC_PRIVATE_DATA_FROM_IPSEC.

Referenced by IpSec4DriverBindingStart(), and IpSec6DriverBindingStart().

```
EFI_STATUS EFIAPI IpSecStop ( IN EFI_DRIVER_BINDING_PROTOCOL *  This,
                              IN EFI_HANDLE                     ControllerHandle,
                              IN UINTN                          NumberOfChildren,
                              IN EFI_HANDLE *                   ChildHandleBuffer,
                              IN UINT8                          IpVersion
                            )
```

Stop this driver on ControllerHandle. This is the worker function for IpSec4(6)DriverbindingStop.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of a device to stop the driver on.
[in]	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If the number of children is zero, stop the entire bus driver.
[in]	*ChildHandleBuffer*	List of Child Handles to Stop.
[in]	*IpVersion*	IP_VERSION_4 or IP_VERSION_6.

Return values:

EFI_SUCCES	This driver removed ControllerHandle.
other	This driver was not removed from this device.

References IKEV2_SA_SESSION::BySessionTable, IKEV2_SA_SESSION_BY_SESSION, _IPSEC_PRIVATE_DATA::Ikev2EstablishedList, Ikev2SaSessionFree(), _IPSEC_PRIVATE_DATA::Ikev2SessionList, IKE_UDP_SERVICE::Input, IPSEC_PRIVATE_DATA_FROM_IPSEC, IPSEC_UDP_SERVICE_FROM_LIST, IKE_UDP_SERVICE::List, IKE_UDP_SERVICE::Output, _IPSEC_PRIVATE_DATA::Udp4List, _IPSEC_PRIVATE_DATA::Udp4Num, _IPSEC_PRIVATE_DATA::Udp6List, and _IPSEC_PRIVATE_DATA::Udp6Num.

Referenced by IpSec4DriverBindingStop(), and IpSec6DriverBindingStop().

```
EFI_STATUS EFIAPI IpSecSupported ( IN EFI_DRIVER_BINDING_PROTOCOL *  This,
                                   IN EFI_HANDLE                     ControllerHandle,
                                   IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath,
                                   IN UINT8                          IpVersion
                                 )
```

Test to see if this driver supports ControllerHandle. This is the worker function for IpSec4(6)DriverbindingSupported.

Parameters:

[in]	*This*	Protocol instance pointer.

[in]	*ControllerHandle*	Handle of device to test.
[in]	*RemainingDevicePath*	Optional parameter used to pick a specific child device to start.
[in]	*IpVersion*	IP_VERSION_4 or IP_VERSION_6.

Return values:

EFI_SUCCES	This driver supports this device.
EFI_ALREADY_STARTED	This driver is already running on this device.
other	This driver does not support this device.

Referenced by IpSec4DriverBindingSupported(), and IpSec6DriverBindingSupported().

Variable Documentation

EFI_DRIVER_BINDING_PROTOCOL gIpSec4DriverBinding

Initial value:

```
{
 IpSec4DriverBindingSupported,
 IpSec4DriverBindingStart,
 IpSec4DriverBindingStop,
 0xa,
 NULL,
 NULL
}
```

Referenced by IpSecDriverEntryPoint().

EFI_DRIVER_BINDING_PROTOCOL gIpSec6DriverBinding

Initial value:

```
{
 IpSec6DriverBindingSupported,
 IpSec6DriverBindingStart,
 IpSec6DriverBindingStop,
 0xa,
 NULL,
 NULL
}
```

Referenced by IpSecDriverEntryPoint().

NetworkPkg/IpSecDxe/IpSecImpl.c File Reference

Functions

BOOLEAN IpSecValidAddressRange (IN UINT8 IpVersion, IN EFI_IP_ADDRESS *Address, IN UINT8 PrefixLength)

VOID IpSecExtractAddressRange (IN EFI_IP_ADDRESS *Address, IN UINT8 PrefixLength, OUT EFI_IP_ADDRESS *Range)

BOOLEAN IpSecMatchIpAddress (IN UINT8 IpVersion, IN EFI_IP_ADDRESS *IpAddr, IN EFI_IP_ADDRESS_INFO *AddressInfo, IN UINT32 AddressCount)

BOOLEAN IpSecMatchNextLayerProtocol (IN UINT8 Protocol, IN UINT8 *IpPayload, IN UINT16 SpdProtocol, IN UINT16 SpdLocalPort, IN UINT16 SpdRemotePort, IN BOOLEAN IsOutbound)

IPSEC_SAD_ENTRY * IpSecLookupSadBySpd (IN LIST_ENTRY *SadList, IN EFI_IP_ADDRESS *DestAddress, IN UINT8 IpVersion)

IPSEC_SAD_ENTRY * IpSecLookupSadBySpi (IN UINT32 Spi, IN EFI_IP_ADDRESS *DestAddress, IN UINT8 IpVersion)

EFI_STATUS IpSecLookupSadEntry (IN IPSEC_PRIVATE_DATA *Private, IN EFI_HANDLE NicHandle, IN UINT8 IpVersion, IN VOID *IpHead, IN UINT8 *IpPayload, IN UINT8 OldLastHead, IN IPSEC_SPD_ENTRY *SpdEntry, OUT IPSEC_SAD_ENTRY **SadEntry)

IPSEC_PAD_ENTRY * IpSecLookupPadEntry (IN UINT8 IpVersion, IN EFI_IP_ADDRESS *IpAddr)

EFI_STATUS IpSecLookupSpdEntry (IN IPSEC_SPD_ENTRY *SpdEntry, IN UINT8 IpVersion, IN VOID *IpHead, IN UINT8 *IpPayload, IN UINT8 Protocol, IN BOOLEAN IsOutbound, OUT EFI_IPSEC_ACTION *Action)

VOID EFIAPI IpSecOnRecyclePacket (IN VOID *Arg)

VOID EFIAPI IpSecRecycleCallback (IN EFI_EVENT Event, IN VOID *Context)

UINT16 IpSecGetPlainExtHeadSize (IN VOID *IpHead, IN UINT8 *LastHead)

EFI_STATUS IpSecEspAuthVerifyPayload (IN UINT8 *EspBuffer, IN UINTN EspSize, IN IPSEC_SAD_ENTRY *SadEntry, IN UINTN IcvSize)

IPSEC_SAD_ENTRY * IpSecFoundSadFromInboundPacket (UINT8 *IpHead, UINT8 IpVersion, UINT32 Spi)

BOOLEAN IpSecIsIp6ExtsValid (IN UINT8 *NextHeader, IN UINT8 *ExtHdrs, IN UINT32 ExtHdrsLen, OUT UINT8 **LastHeader, OUT UINT32 *RealExtsLen)

VOID IpSecTunnelInboundPacket (IN OUT UINT8 *IpHead, IN UINT8 *IpPayload, IN UINT8 IpVersion, IN IPSEC_SAD_DATA *SadData, IN OUT UINT8 *LastHead)

UINT8 * IpSecTunnelOutboundPacket (IN OUT UINT8 *IpHead, IN UINT8 IpVersion, IN IPSEC_SAD_DATA *SadData, IN OUT UINT8 *LastHead, IN VOID **OptionsBuffer, IN UINT32 *OptionsLength, IN OUT EFI_IPSEC_FRAGMENT_DATA **FragmentTable, IN UINT32 *FragmentCount)

EFI_STATUS IpSecEspInboundPacket (IN UINT8 IpVersion, IN OUT VOID *IpHead, OUT UINT8 *LastHead, IN OUT VOID **OptionsBuffer, IN OUT UINT32 *OptionsLength, IN OUT EFI_IPSEC_FRAGMENT_DATA **FragmentTable, IN OUT UINT32 *FragmentCount, OUT EFI_IPSEC_SPD_SELECTOR **SpdSelector, OUT EFI_EVENT *RecycleEvent)

EFI_STATUS IpSecEspOutboundPacket (IN UINT8 IpVersion, IN OUT VOID *IpHead, IN OUT UINT8 *LastHead, IN OUT VOID **OptionsBuffer, IN OUT UINT32 *OptionsLength, IN OUT EFI_IPSEC_FRAGMENT_DATA **FragmentTable, IN OUT UINT32 *FragmentCount, IN IPSEC_SAD_ENTRY *SadEntry, OUT EFI_EVENT *RecycleEvent)

EFI_STATUS IpSecProtectInboundPacket (IN UINT8 IpVersion, IN OUT VOID *IpHead, IN OUT UINT8 *LastHead, IN OUT VOID **OptionsBuffer, IN OUT UINT32

*OptionsLength, IN OUT EFI_IPSEC_FRAGMENT_DATA **FragmentTable, IN OUT UINT32 *FragmentCount, OUT EFI_IPSEC_SPD_SELECTOR **SpdEntry, OUT EFI_EVENT *RecycleEvent)

EFI_STATUS IpSecProtectOutboundPacket (IN UINT8 IpVersion, IN OUT VOID *IpHead, IN OUT UINT8 *LastHead, IN OUT VOID **OptionsBuffer, IN OUT UINT32 *OptionsLength, IN OUT EFI_IPSEC_FRAGMENT_DATA **FragmentTable, IN OUT UINT32 *FragmentCount, IN IPSEC_SAD_ENTRY *SadEntry, OUT EFI_EVENT *RecycleEvent)

Detailed Description

The implementation of IPsec.

(C) Copyright 2015 Hewlett-Packard Development Company, L.P.
Copyright (c) 2009 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS IpSecEspAuthVerifyPayload ( IN UINT8 *                EspBuffer,
                                       IN UINTN                  EspSize,
                                       IN IPSEC_SAD_ENTRY *      SadEntry,
                                       IN UINTN                  IcvSize
                                     )
```

Verify if the Authentication payload is correct.

Parameters:

[in]	*EspBuffer*	Points to the ESP wrapped buffer.
[in]	*EspSize*	The size of the ESP wrapped buffer.
[in]	*SadEntry*	The related SAD entry to store the authentication algorithm key.
[in]	*IcvSize*	The length of ICV.

Return values:

EFI_SUCCESS	The authentication data is correct.
EFI_ACCESS_DENIED	The authentication data is not correct.

References HASH_DATA_FRAGMENT::Data, HASH_DATA_FRAGMENT::DataSize, and IpSecCryptoIoHmac().

Referenced by IpSecEspInboundPacket().

```
EFI_STATUS IpSecEspInboundPacket ( IN UINT8                                IpVersion,
```

	IN OUT VOID *	*IpHead*,
	OUT UINT8 *	*LastHead*,
	IN OUT VOID **	*OptionsBuffer*,
	IN OUT UINT32 *	*OptionsLength*,
	IN OUT EFI_IPSEC_FRAGMENT_DATA **	*FragmentTable*,
	IN OUT UINT32 *	*FragmentCount*,
	OUT EFI_IPSEC_SPD_SELECTOR **	*SpdSelector*,
	OUT EFI_EVENT *	*RecycleEvent*
)		

The actual entry to relative function processes the inbound traffic of ESP header.

This function is the subfunction of IpSecProtectInboundPacket(). It checks the received packet security property and trim the ESP header and then returns without an IPsec protected IP Header and FramgmentTable.

Parameters:

`[in]`	*IpVersion*	The version of IP.
`[in,out]`	*IpHead*	Points to the IP header containing the ESP header to be trimed on input, and without ESP header on return.
`[out]`	*LastHead*	The Last Header in IP header on return.
`[in,out]`	*OptionsBuffer*	Pointer to the options buffer.
`[in,out]`	*OptionsLength*	Length of the options buffer.
`[in,out]`	*FragmentTable*	Pointer to a list of fragments in the form of IPsec protected on input, and without IPsec protected on return.
`[in,out]`	*FragmentCount*	The number of fragments.
`[out]`	*SpdSelector*	Pointer to contain the address of SPD selector on return.
`[out]`	*RecycleEvent*	The event for recycling of resources.

Return values:

EFI_SUCCESS	The operation was successful.
EFI_ACCESS_DENIED	One or more following conditions is TRUE: ◊ ESP header was not found or mal-format. ◊ The related SAD entry was not found. ◊ The related SAD entry does not support the ESP protocol.
EFI_OUT_OF_RESOURCES	The required system resource can't be allocated.

References _IPSEC_SAD_DATA::AlgoInfo, _IPSEC_SAD_ENTRY::Data, _IPSEC_RECYCLE_CONTEXT::FragmentTable, _IPSEC_SAD_ENTRY::Id, IpSecCryptoIoDecrypt(), IpSecEspAuthVerifyPayload(), IpSecFoundSadFromInboundPacket(), IpSecGetEncryptBlockSize(), IpSecGetEncryptIvLength(), IpSecGetIcvLength(), IpSecGetPlainExtHeadSize(), IpSecOnRecyclePacket(), IpSecRecycleCallback(), IpSecTunnelInboundPacket(), _IPSEC_SAD_DATA::ManualSet, _IPSEC_SAD_DATA::Mode, _EFI_ESP_TAIL::NextHeader, _EFI_ESP_TAIL::PaddingLength, _IPSEC_RECYCLE_CONTEXT::PayloadBuffer, _IPSEC_SAD_DATA::SpdSelector, and _EFI_ESP_HEADER::Spi.

Referenced by IpSecProtectInboundPacket().

EFI_STATUS IpSecEspOutboundPacket	(IN UINT8	*IpVersion*,

```
                                IN OUT VOID *                          IpHead,
                                IN OUT UINT8 *                         LastHead,
                                IN OUT VOID **                         OptionsBuffer,
                                IN OUT UINT32 *                        OptionsLength,
                                IN OUT EFI_IPSEC_FRAGMENT_DATA **      FragmentTable,
                                IN OUT UINT32 *                        FragmentCount,
                                IN IPSEC_SAD_ENTRY *                   SadEntry,
                                OUT EFI_EVENT *                        RecycleEvent
                              )
```

The actual entry to the relative function processes the output traffic using the ESP protocol.

This function is the subfunction of IpSecProtectOutboundPacket(). It protected the sending packet by encrypting its payload and inserting ESP header in the orginal IP header, then return the IpHeader and IPsec protected Fragmentable.

Parameters:

[in]	*IpVersion*	The version of IP.
[in,out]	*IpHead*	Points to IP header containing the orginal IP header to be processed on input, and inserted ESP header on return.
[in,out]	*LastHead*	The Last Header in IP header.
[in,out]	*OptionsBuffer*	Pointer to the options buffer.
[in,out]	*OptionsLength*	Length of the options buffer.
[in,out]	*FragmentTable*	Pointer to a list of fragments to be protected by IPsec on input, and with IPsec protected on return.
[in,out]	*FragmentCount*	The number of fragments.
[in]	*SadEntry*	The related SAD entry.
[out]	*RecycleEvent*	The event for recycling of resources.

Return values:

EFI_SUCCESS	The operation was successful.
EFI_OUT_OF_RESOURCES	The required system resources can't be allocated.

References _IPSEC_SAD_DATA::AlgoInfo, HASH_DATA_FRAGMENT::Data, HASH_DATA_FRAGMENT::DataSize, _IPSEC_RECYCLE_CONTEXT::FragmentTable, IPSEC_ESP_PROTOCOL, IpSecCryptoIoEncrypt(), IpSecCryptoIoHmac(), IpSecGenerateIv(), IpSecGetEncryptBlockSize(), IpSecGetEncryptIvLength(), IpSecGetIcvLength(), IpSecGetPlainExtHeadSize(), IpSecRecycleCallback(), IpSecTunnelOutboundPacket(), _IPSEC_SAD_DATA::ManualSet, _IPSEC_SAD_DATA::Mode, _EFI_ESP_TAIL::NextHeader, _EFI_ESP_TAIL::PaddingLength, _IPSEC_RECYCLE_CONTEXT::PayloadBuffer, _IPSEC_SAD_DATA::SequenceNumber, _EFI_ESP_HEADER::SequenceNumber, _EFI_ESP_HEADER::Spi, _IPSEC_SAD_DATA::TunnelDestAddress, and _IPSEC_SAD_DATA::TunnelSourceAddress.

Referenced by IpSecProtectOutboundPacket().

```
VOID IpSecExtractAddressRange ( IN EFI_IP_ADDRESS *    Address,
                                IN UINT8               PrefixLength,
                                OUT EFI_IP_ADDRESS *   Range
                              )
```

Extrct the Address Range from a Address.

This function keep the prefix address and zero other part address.

Parameters:

[in]	*Address*	Point to a specified address.
[in]	*PrefixLength*	The prefix length.
[out]	*Range*	Contain the return Address Range.

Referenced by IpSecMatchIpAddress().

```
IPSEC_SAD_ENTRY* IpSecFoundSadFromInboundPacket ( UINT8 *   IpHead,
                                                  UINT8     IpVersion,
                                                  UINT32    Spi
                                                )
```

Search the related SAD entry by the input .

Parameters:

[in]	*IpHead*	The pointer to IP header.
[in]	*IpVersion*	The version of IP (IP4 or IP6).
[in]	*Spi*	The SPI used to search the related SAD entry.

Return values:

NULL	Not find the related SAD entry.
IPSEC_SAD_ENTRY	Return the related SAD entry.

References IpSecLookupSadBySpi().

Referenced by IpSecEspInboundPacket().

```
UINT16 IpSecGetPlainExtHeadSize ( IN VOID *    IpHead,
                                  IN UINT8 *   LastHead
                                )
```

Calculate the extension hader of IP. The return length only doesn't contain the fixed IP header length.

Parameters:

[in]	*IpHead*	Points to an IP head to be calculated.
[in]	*LastHead*	Points to the last header of the IP header.

Returns:

The length of the extension header.

Referenced by IpSecEspInboundPacket(), and IpSecEspOutboundPacket().

```
BOOLEAN IpSecIsIp6ExtsValid ( IN UINT8 *       NextHeader,
                              IN UINT8 *       ExtHdrs,
                              IN UINT32        ExtHdrsLen,
                              OUT UINT8 **     LastHeader,
```

```
                                    OUT UINT32 *   RealExtsLen
                                  )
```

Validate the IP6 extension header format for both the packets we received and that we will transmit.

Parameters:

[in]	*NextHeader*	The next header field in IPv6 basic header.
[in]	*ExtHdrs*	The first bye of the option.
[in]	*ExtHdrsLen*	The length of the whole option.
[out]	*LastHeader*	The pointer of NextHeader of the last extension header processed by IP6.
[out]	*RealExtsLen*	The length of extension headers processed by IP6 layer. This is an optional parameter that may be NULL.

Return values:

TRUE	The option is properly formated.
FALSE	The option is malformated.

Referenced by IpSecTunnelInboundPacket().

```
IPSEC_PAD_ENTRY* IpSecLookupPadEntry   (  IN UINT8              IpVersion,
                                          IN
                                          EFI_IP_ADDRESS        IpAddr
                                          *
                                       )
```

Find a PAD entry according to a remote IP address.

Parameters:

[in]	*IpVersion*	The version of IP.
[in]	*IpAddr*	Points to remote IP address.

Returns:

the pointer of related PAD entry.

References _IPSEC_PAD_ENTRY::Id, IPSEC_PAD_ENTRY_FROM_LIST, IpSecMatchIpAddress(), and mConfigData.

Referenced by IkeNegotiate(), and Ikev2HandleSa().

```
IPSEC_SAD_ENTRY* IpSecLookupSadBySpd  ( IN LIST_ENTRY *          SadList,
                                        IN EFI_IP_ADDRESS *      DestAddress,
                                        IN UINT8                 IpVersion
                                      )
```

Find the SAD through a specified SPD's SAD list.

Parameters:

[in]	*SadList*	SAD list related to a specified SPD entry.
[in]	*DestAddress*	The destination address used to find the SAD entry.
[in]	*IpVersion*	The IP version. Ip4 or Ip6.

Returns:

The pointer to a certain SAD entry.

References _IPSEC_SAD_ENTRY::Data, IPSEC_SAD_ENTRY_FROM_SPD, IpSecMatchIpAddress(), and _IPSEC_SAD_DATA::SpdSelector.

Referenced by IpSecLookupSadEntry().

```
IPSEC_SAD_ENTRY* IpSecLookupSadBySpi ( IN UINT32              Spi,
                                       IN EFI_IP_ADDRESS *    DestAddress,
                                       IN UINT8               IpVersion
                                     )
```

Find the SAD through whole SAD list.

Parameters:

[in]	*Spi*	The SPI used to search the SAD entry.
[in]	*DestAddress*	The destination used to search the SAD entry.
[in]	*IpVersion*	The IP version. Ip4 or Ip6.

Returns:

the pointer to a certain SAD entry.

References _IPSEC_SAD_ENTRY::Data, _IPSEC_SAD_ENTRY::Id, IPSEC_SAD_ENTRY_FROM_LIST, IpSecMatchIpAddress(), mConfigData, _IPSEC_SAD_DATA::Mode, _IPSEC_SAD_DATA::SpdSelector, and _IPSEC_SAD_DATA::TunnelDestAddress.

Referenced by IpSecFoundSadFromInboundPacket().

```
EFI_STATUS IpSecLookupSadEntry ( IN IPSEC_PRIVATE_DATA *    Private,
                                 IN EFI_HANDLE              NicHandle,
                                 IN UINT8                   IpVersion,
                                 IN VOID *                  IpHead,
                                 IN UINT8 *                 IpPayload,
                                 IN UINT8                   OldLastHead,
                                 IN IPSEC_SPD_ENTRY *       SpdEntry,
                                 OUT IPSEC_SAD_ENTRY **     SadEntry
                               )
```

Look up if there is existing SAD entry for specified IP packet sending.

This function is called by the IPsecProcess when there is some IP packet needed to send out. This function checks if there is an existing SAD entry that can be serviced to this IP packet sending. If no existing SAD entry could be used, this function will invoke an IPsec Key Exchange Negotiation.

Parameters:

[in]	*Private*	Points to private data.
[in]	*NicHandle*	Points to a NIC handle.
[in]	*IpVersion*	The version of IP.
[in]	*IpHead*	The IP Header of packet to be sent out.
[in]	*IpPayload*	The IP Payload to be sent out.
[in]	*OldLastHead*	The Last protocol of the IP packet.
[in]	*SpdEntry*	Points to a related SPD entry.
[out]	*SadEntry*	Contains the Point of a related SAD entry.

Return values:

EFI_DEVICE_ERROR	One of following conditions is TRUE: ◊ If don't find related UDP service. ◊ Sequence Number is used up. ◊ Extension Sequence Number is used up.
EFI_NOT_READY	No existing SAD entry could be used.
EFI_SUCCESS	Find the related SAD entry.

References _IPSEC_SAD_ENTRY::Data, _IPSEC_SAD_DATA::ESNEnabled, IkeLookupUdp(), IkeNegotiate(), IpSecLookupSadBySpd(), _IPSEC_SAD_DATA::ManualSet, and _IPSEC_SAD_DATA::SequenceNumber.

Referenced by IpSecProcess().

```
EFI_STATUS IpSecLookupSpdEntry ( IN IPSEC_SPD_ENTRY *      SpdEntry,
                                 IN UINT8                  IpVersion,
                                 IN VOID *                 IpHead,
                                 IN UINT8 *                IpPayload,
                                 IN UINT8                  Protocol,
                                 IN BOOLEAN                IsOutbound,
                                 OUT EFI_IPSEC_ACTION *    Action
                               )
```

Check if the specified IP packet can be serviced by this SPD entry.

Parameters:

[in]	*SpdEntry*	Point to SPD entry.
[in]	*IpVersion*	Version of IP.
[in]	*IpHead*	Point to IP header.
[in]	*IpPayload*	Point to IP payload.
[in]	*Protocol*	The Last protocol of IP packet.
[in]	*IsOutbound*	Traffic direction.
[out]	*Action*	The support action of SPD entry.

Return values:

EFI_SUCCESS	Find the related SPD.
EFI_NOT_FOUND	Not find the related SPD entry;

References IpSecMatchIpAddress(), and IpSecMatchNextLayerProtocol().

Referenced by IpSecProcess().

```
BOOLEAN IpSecMatchIpAddress ( IN UINT8                    IpVersion,
                              IN EFI_IP_ADDRESS *         IpAddr,
                              IN EFI_IP_ADDRESS_INFO *    AddressInfo,
                              IN UINT32                   AddressCount
                            )
```

Checks if the IP Address in the address range of AddressInfos specified.

Parameters:

[in]	*IpVersion*	The IP version.
[in]	*IpAddr*	Point to EFI_IP_ADDRESS to be check.
[in]	*AddressInfo*	A list of EFI_IP_ADDRESS_INFO that is used to check the IP Address is matched.
[in]	*AddressCount*	The total numbers of the AddressInfo.

Return values:

TRUE	If the Specified IP Address is in the range of the AddressInfos specified.
FALSE	If the Specified IP Address is not in the range of the AddressInfos specified.

References IpSecExtractAddressRange(), and IpSecValidAddressRange().

Referenced by Ikev2MatchSpdEntry(), IpSecLookupPadEntry(), IpSecLookupSadBySpd(), IpSecLookupSadBySpi(), and IpSecLookupSpdEntry().

```
BOOLEAN IpSecMatchNextLayerProtocol ( IN UINT8      Protocol,
                                      IN UINT8 *    IpPayload,
                                      IN UINT16     SpdProtocol,
                                      IN UINT16     SpdLocalPort,
                                      IN UINT16     SpdRemotePort,
                                      IN BOOLEAN    IsOutbound
                                    )
```

Check if the specified Protocol and Prot is supported by the specified SPD Entry.

This function is the subfunction of IPsecLookUpSpdEntry() that is used to check if the sent/received IKE packet has the related SPD entry support.

Parameters:

[in]	*Protocol*	The Protocol to be checked.
[in]	*IpPayload*	Point to IP Payload to be check.
[in]	*SpdProtocol*	The Protocol supported by SPD.
[in]	*SpdLocalPort*	The Local Port in SPD.
[in]	*SpdRemotePort*	The Remote Port in SPD.
[in]	*IsOutbound*	Flag to indicate the is for IKE Packet sending or recieving.

Return values:

TRUE	The Protocol and Port are supported by the SPD Entry.
FALSE	The Protocol and Port are not supported by the SPD Entry.

References EFI_IPSEC_ANY_PROTOCOL.

Referenced by IpSecLookupSpdEntry().

```
VOID EFIAPI IpSecOnRecyclePacket ( IN VOID * Arg )
```

The call back function of NetbufFromExt.

Parameters:

[in]	*Arg*	The argument passed from the caller.

Referenced by IpSecEspInboundPacket(), and IpSecTunnelOutboundPacket().

```
EFI_STATUS IpSecProtectInboundPacket (   IN UINT8                          IpVersion,
                                         IN OUT VOID *                     IpHead,
                                         IN OUT UINT8 *                    LastHead,
                                         IN OUT VOID **                    OptionsBuffer,
                                         IN OUT UINT32 *                   OptionsLength,
                                         IN OUT
                                         EFI_IPSEC_FRAGMENT_DATA           FragmentTable,
                                         **
                                         IN OUT UINT32 *                   FragmentCount,
                                         OUT EFI_IPSEC_SPD_SELECTOR        SpdEntry,
                                         **
                                         OUT EFI_EVENT *                   RecycleEvent
                                       )
```

This function processes the inbound traffic with IPsec.

It checks the received packet security property, trims the ESP/AH header, and then returns without an IPsec protected IP Header and FragmentTable.

Parameters:

[in]	*IpVersion*	The version of IP.
[in,out]	*IpHead*	Points to IP header containing the ESP/AH header to be trimed on input, and without ESP/AH header on return.
[in,out]	*LastHead*	The Last Header in IP header on return.
[in,out]	*OptionsBuffer*	Pointer to the options buffer.
[in,out]	*OptionsLength*	Length of the options buffer.
[in,out]	*FragmentTable*	Pointer to a list of fragments in form of IPsec protected on input, and without IPsec protected on return.
[in,out]	*FragmentCount*	The number of fragments.
[out]	*SpdEntry*	Pointer to contain the address of SPD entry on return.
[out]	*RecycleEvent*	The event for recycling of resources.

Return values:

EFI_SUCCESS	The operation was successful.
EFI_UNSUPPORTED	The IPSEC protocol is not supported.

References IPSEC_ESP_PROTOCOL, and IpSecEspInboundPacket().

Referenced by IpSecProcess().

```
EFI_STATUS
IpSecProtectOutboundPacket   ( IN UINT8                          IpVersion,
                               IN OUT VOID *                     IpHead,
                               IN OUT UINT8 *                    LastHead,
                               IN OUT VOID **                    OptionsBuffer,
                               IN OUT UINT32 *                   OptionsLength,
                                                                 FragmentTable,
```

```
                                    IN OUT EFI_IPSEC_FRAGMENT_DATA
                                    **
                                    IN OUT UINT32 *                 FragmentCount,
                                    IN IPSEC_SAD_ENTRY *            SadEntry,
                                    OUT EFI_EVENT *                 RecycleEvent
                                  )
```

This fucntion processes the output traffic with IPsec.

It protected the sending packet by encrypting it payload and inserting ESP/AH header in the orginal IP header, then return the IpHeader and IPsec protected Fragmentable.

Parameters:

[in]	*IpVersion*	The version of IP.
[in,out]	*IpHead*	Point to IP header containing the orginal IP header to be processed on input, and inserted ESP/AH header on return.
[in,out]	*LastHead*	The Last Header in IP header.
[in,out]	*OptionsBuffer*	Pointer to the options buffer.
[in,out]	*OptionsLength*	Length of the options buffer.
[in,out]	*FragmentTable*	Pointer to a list of fragments to be protected by IPsec on input, and with IPsec protected on return.
[in,out]	*FragmentCount*	Number of fragments.
[in]	*SadEntry*	Related SAD entry.
[out]	*RecycleEvent*	Event for recycling of resources.

Return values:

EFI_SUCCESS	The operation is successful.
EFI_UNSUPPORTED	If the IPSEC protocol is not supported.

References IpSecEspOutboundPacket().

Referenced by IpSecProcess().

```
VOID EFIAPI IpSecRecycleCallback  ( IN EFI_EVENT   Event,
                                    IN VOID *      Context
                                  )
```

This is a Notification function. It is called when the related IP6_TXTOKEN_WRAP is released.

Parameters:

[in]	*Event*	The related event.
[in]	*Context*	The data passed by the caller.

References _IPSEC_RECYCLE_CONTEXT::FragmentTable, and _IPSEC_RECYCLE_CONTEXT::PayloadBuffer.

Referenced by IpSecEspInboundPacket(), and IpSecEspOutboundPacket().

```
VOID IpSecTunnelInboundPacket  ( IN OUT UINT8 *        IpHead,
                                 IN UINT8 *            IpPayload,
                                 IN UINT8              IpVersion,
```

```
IN IPSEC_SAD_DATA *    SadData,
IN OUT UINT8 *         LastHead
)
```

The actual entry to process the tunnel header and inner header for tunnel mode outbound traffic.

This function is the subfunction of IpSecEspInboundPacket(). It change the destination Ip address to the station address and recalculate the uplayyer's checksum.

Parameters:

[in,out]	*IpHead*	Points to the IP header containing the ESP header to be trimed on input, and without ESP header on return.
[in]	*IpPayload*	The decrypted Ip payload. It start from the inner header.
[in]	*IpVersion*	The version of IP.
[in]	*SadData*	Pointer of the relevant SAD.
[in,out]	*LastHead*	The Last Header in IP header on return.

References IpSecIsIp6ExtsValid().

Referenced by IpSecEspInboundPacket().

```
UINT8* IpSecTunnelOutboundPacket ( IN OUT UINT8 *                    IpHead,
                                   IN UINT8                          IpVersion,
                                   IN IPSEC_SAD_DATA *               SadData,
                                   IN OUT UINT8 *                    LastHead,
                                   IN VOID **                        OptionsBuffer,
                                   IN UINT32 *                       OptionsLength,
                                   IN OUT EFI_IPSEC_FRAGMENT_DATA ** FragmentTable,
                                   IN UINT32 *                       FragmentCount
                                 )
```

The actual entry to create inner header for tunnel mode inbound traffic.

This function is the subfunction of IpSecEspOutboundPacket(). It create the sending packet by encrypting its payload and inserting ESP header in the orginal IP header, then return the IpHeader and IPsec protected Fragmentable.

Parameters:

[in,out]	*IpHead*	Points to IP header containing the orginal IP header to be processed on input, and inserted ESP header on return.
[in]	*IpVersion*	The version of IP.
[in]	*SadData*	The related SAD data.
[in,out]	*LastHead*	The Last Header in IP header.
[in]	*OptionsBuffer*	Pointer to the options buffer.
[in]	*OptionsLength*	Length of the options buffer.
[in,out]	*FragmentTable*	Pointer to a list of fragments to be protected by IPsec on input, and with IPsec protected on return.
[in]	*FragmentCount*	The number of fragments.

Return values:

EFI_SUCCESS	The operation was successful.
EFI_OUT_OF_RESOURCES	The required system resources can't be allocated.

References IpSecOnRecyclePacket().

Referenced by IpSecEspOutboundPacket().

```
BOOLEAN IpSecValidAddressRange ( IN UINT8              IpVersion,
                                 IN EFI_IP_ADDRESS *   Address,
                                 IN UINT8              PrefixLength
                               )
```

Check if the specified Address is the Valid Address Range.

This function checks if the bytes after prefixed length are all Zero in this Address. This Address is supposed to point to a range address. That means it should gives the correct prefixed address and the bytes outside the prefixed are zero.

Parameters:

[in]	*IpVersion*	The IP version.
[in]	*Address*	Points to EFI_IP_ADDRESS to be checked.
[in]	*PrefixLength*	The PrefixeLength of this address.

Return values:

TRUE	The address is a vaild address range.
FALSE	The address is not a vaild address range.

Referenced by IpSecMatchIpAddress().

NetworkPkg/IpSecDxe/IpSecImpl.h File Reference

Data Structures

struct _EF
struct _EF
struct _EF
struct _IPS
struct _IPS
struct _IPS
struct _IPS
struct _IPS
struct _IPS
struct HAS
struct _IPS

Defines

"memItemRight" valign="bottom">IPSEC_PRIVATE_DATA_SIGNATURE SIGNATURE_32 ('I', 'P', 'S', 'E')
#define IPSE
(a, I
IPSE
#define IPSE
(a, I
IPSE
#define IPSE
(a, I
IPSE
#define IPSE
(a, I
#define IPSE
(a, I
#define IPSE
(a, I
#define IPSE
(a, I
#define IPSE
(a, I
#define IPSE
#define IPSE
#define IPSE
#define IPSE
#define IPSE

Typedefs

typedef struct _IPSEC_PRIVATE_DATA IPSE
typedef struct _IPSEC_SPD_ENTRY IPSE
typedef struct _IPSEC_PAD_ENTRY IPSE
typedef struct _IPSEC_SPD_DATA IPSE
typedef struct _EFI_AH_HEADER EFI_

typedef struct _EFI_ESP_HEADER EFI_ESP_H
typedef struct _EFI_ESP_TAIL EFI_ESP_T
typedef struct _IPSEC_SAD_DATA IPSEC_SAD
typedef struct _IPSEC_SAD_ENTRY IPSEC_SAD
typedef struct
_IPSEC_RECYCLE_CONTEXT IPSEC_REC

Functions

EFI_STATUS IpSecProtect
IpVersion, I
UINT8 *Las
**OptionsBu
*OptionsLen
EFI_IPSEC_
**Fragment
*FragmentC
EFI_IPSEC_
OUT EFI_E
EFI_STATUS IpSecProtect
IpVersion, I
UINT8 *Las
**OptionsBu
*OptionsLen
EFI_IPSEC_
**Fragment
*FragmentC
*SadEntry, C
BOOLEAN IpSecMatchI
IN EFI_IP_A
EFI_IP_ADD
UINT32 Ad
IPSEC_PAD_ENTRY * IpSecLooku
IN EFI_IP_A
EFI_STATUS IpSecLooku
IPSEC_SPD
IpVersion, I
*IpPayload,
BOOLEAN
EFI_IPSEC_
EFI_STATUS IpSecLooku
IPSEC_PRI
EFI_HAND
IpVersion, I
*IpPayload,
IPSEC_SPD
IPSEC_SAD
IPSEC_SAD_ENTRY * IpSecLooku
EFI_IP_ADD
IpVersion)
EFI_STATUS EFIAPI IpSecProces
*This, IN EF
UINT8 IpVe
OUT UINT8

**O
*Op
EFI_
**Fr
*Fra
EFI_
OUT

Variables

EFI_DPC_PROTOCOL * mDp
EFI_IPSEC2_PROTOCOL mIp
EFI_COMPONENT_NAME2_PROTOCOL gIpS
EFI_COMPONENT_NAME_PROTOCOL gIpS

Detailed Description

The definitions related to IPsec protocol implementation.

Copyright (c) 2009 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

#define IPSEC_DEFAULT_VARIABLE_SIZE 0x100

Referenced by IpSecConfigSave().

#define IPSEC_ESP_PROTOCOL 50

Referenced by IpSecEspOutboundPacket(), and IpSecProtectInboundPacket().

#define IPSEC_PAD_ENTRY_FROM_LIST (a) BASE_CR (a, IPSEC_PAD_ENTRY, List)

Referenced by GetPadEntry(), IpSecLookupPadEntry(), and SetPadEntry().

#define IPSEC_PRIVATE_DATA_FROM_IPSEC (a) CR (a, IPSEC_PRIVATE_DATA, IpSec, IPSEC_PRIVATE_DATA_SIGNATURE)

Referenced by IpSecProcess(), IpSecStart(), and IpSecStop().

#define IPSEC_PRIVATE_DATA_FROM_UDP4LIST (a) CR (a, IPSEC_PRIVATE_DATA, Udp4List, IPSEC_PRIVATE_DATA_SIGNATURE)

Referenced by IkeDispatch(), IkeNegotiate(), Ikev2HandleChildSa(), Ikev2HandleInfo(), Ikev2HandleSa(), and Ikev2NegotiateSa().

#define IPSEC_PRIVATE_DATA_FROM_UDP6LIST (a) CR (a, IPSEC_PRIVATE_DATA, Udp6List, IPSEC_PRIVATE_DATA_SIGNATURE)

Referenced by IkeDispatch(), IkeNegotiate(), Ikev2HandleChildSa(), Ikev2HandleInfo(), Ikev2HandleSa(), and Ikev2NegotiateSa().

#define IPSEC_PRIVATE_DATA_SIGNATURE SIGNATURE_32 ('I', 'P', 'S', 'E')

Referenced by IpSecDriverEntryPoint().

#define IPSEC_SAD_ENTRY_FROM_LIST (a) BASE_CR (a, IPSEC_SAD_ENTRY, List)

Referenced by GetSadEntry(), IpSecLookupSadBySpi(), SetSadEntry(), and SetSpdEntry().

#define IPSEC_SAD_ENTRY_FROM_SPD (a) BASE_CR (a, IPSEC_SAD_ENTRY, BySpd)

Referenced by GetSpdEntry(), IpSecLookupSadBySpd(), and SetSpdEntry().

#define IPSEC_SPD_ENTRY_FROM_LIST (a) BASE_CR (a, IPSEC_SPD_ENTRY, List)

Referenced by GetSpdEntry(), IkeSearchSpdEntry(), IpSecProcess(), SetSadEntry(), and SetSpdEntry().

#define IPSEC_STATUS_DISABLED 0

#define IPSEC_STATUS_ENABLED 1

#define IPSEC_UDP_SERVICE_FROM_LIST (a) BASE_CR (a, IKE_UDP_SERVICE, List)

Referenced by IkeLookupUdp(), and IpSecStop().

Typedef Documentation

typedef struct _EFI_AH_HEADER EFI_AH_HEADER

typedef struct _EFI_ESP_HEADER EFI_ESP_HEADER

typedef struct _EFI_ESP_TAIL EFI_ESP_TAIL

typedef struct _IPSEC_PAD_ENTRY IPSEC_PAD_ENTRY

typedef struct _IPSEC_PRIVATE_DATA IPSEC_PRIVATE_DATA

typedef struct _IPSEC_RECYCLE_CONTEXT IPSEC_RECYCLE_CONTEXT

typedef struct _IPSEC_SAD_DATA IPSEC_SAD_DATA

typedef struct _IPSEC_SAD_ENTRY IPSEC_SAD_ENTRY

typedef struct _IPSEC_SPD_DATA IPSEC_SPD_DATA

typedef struct _IPSEC_SPD_ENTRY IPSEC_SPD_ENTRY

Function Documentation

```
IPSEC_PAD_ENTRY* IpSecLookupPadEntry ( IN UINT8 IpVersion,
                                       IN EFI_IP_ADDRESS * IpAddr
                                       )
```

Find a PAD entry according to remote IP address.

Parameters:

[in]	*IpVersion*	The version of IP.
[in]	*IpAddr*	Point to remote IP address.

Returns:

The pointer of related PAD entry.

Find a PAD entry according to a remote IP address.

Parameters:

[in]	*IpVersion*	The version of IP.
[in]	*IpAddr*	Points to remote IP address.

Returns:

the pointer of related PAD entry.

References _IPSEC_PAD_ENTRY::Id, IPSEC_PAD_ENTRY_FROM_LIST, IpSecMatchIpAddress(), and mConfigData.

Referenced by IkeNegotiate(), and Ikev2HandleSa().

```
IPSEC_SAD_ENTRY* IpSecLookupSadBySpi ( IN UINT32              Spi,
                                       IN EFI_IP_ADDRESS *    DestAddress,
                                       IN UINT8               IpVersion
                                     )
```

Find the SAD through whole SAD list.

Parameters:

[in]	*Spi*	The SPI used to search the SAD entry.
[in]	*DestAddress*	The destination used to search the SAD entry.
[in]	*IpVersion*	The IP version. Ip4 or Ip6.

Returns:

The pointer to a certain SAD entry.

Find the SAD through whole SAD list.

Parameters:

[in]	*Spi*	The SPI used to search the SAD entry.
[in]	*DestAddress*	The destination used to search the SAD entry.
[in]	*IpVersion*	The IP version. Ip4 or Ip6.

Returns:

the pointer to a certain SAD entry.

References _IPSEC_SAD_ENTRY::Data, _IPSEC_SAD_ENTRY::Id, IPSEC_SAD_ENTRY_FROM_LIST, IpSecMatchIpAddress(), mConfigData, _IPSEC_SAD_DATA::Mode, _IPSEC_SAD_DATA::SpdSelector, and _IPSEC_SAD_DATA::TunnelDestAddress.

Referenced by IpSecFoundSadFromInboundPacket().

```
EFI_STATUS IpSecLookupSadEntry ( IN IPSEC_PRIVATE_DATA *    Private,
                                 IN EFI_HANDLE              NicHandle,
                                 IN UINT8                   IpVersion,
                                 IN VOID *                  IpHead,
                                 IN UINT8 *                 IpPayload,
                                 IN UINT8                   OldLastHead,
                                 IN IPSEC_SPD_ENTRY *       SpdEntry,
                                 OUT IPSEC_SAD_ENTRY **     SadEntry
                               )
```

Look up if there is existing SAD entry for specified IP packet sending.

This function is called by the IPsecProcess when there is some IP packet needed to send out. This function checks if there is an existing SAD entry that can be serviced to this IP packet sending. If no existing SAD entry could be used, this function will invoke an IPsec Key Exchange Negotiation.

Parameters:

[in]	*Private*	Points to private data.
[in]	*NicHandle*	Points to a NIC handle.
[in]	*IpVersion*	The version of IP.
[in]	*IpHead*	The IP Header of packet to be sent out.
[in]	*IpPayload*	The IP Payload to be sent out.
[in]	*OldLastHead*	The Last protocol of the IP packet.
[in]	*SpdEntry*	Points to a related SPD entry.
[out]	*SadEntry*	Contains the Point of a related SAD entry.

Return values:

EFI_DEVICE_ERROR	One of following conditions is TRUE: ◊ If don't find related UDP service. ◊ Sequence Number is used up. ◊ Extension Sequence Number is used up.
EFI_NOT_READY	No existing SAD entry could be used.
EFI_SUCCESS	Find the related SAD entry.

References _IPSEC_SAD_ENTRY::Data, _IPSEC_SAD_DATA::ESNEnabled, IkeLookupUdp(), IkeNegotiate(), IpSecLookupSadBySpd(), _IPSEC_SAD_DATA::ManualSet, and _IPSEC_SAD_DATA::SequenceNumber.

Referenced by IpSecProcess().

```
EFI_STATUS IpSecLookupSpdEntry ( IN IPSEC_SPD_ENTRY *      SpdEntry,
                                 IN UINT8                  IpVersion,
                                 IN VOID *                 IpHead,
                                 IN UINT8 *                IpPayload,
                                 IN UINT8                  Protocol,
                                 IN BOOLEAN                IsOutbound,
                                 OUT EFI_IPSEC_ACTION *    Action
                                 )
```

Check if the specified IP packet can be serviced by this SPD entry.

Parameters:

[in]	*SpdEntry*	Point to SPD entry.
[in]	*IpVersion*	Version of IP.
[in]	*IpHead*	Point to IP header.
[in]	*IpPayload*	Point to IP payload.
[in]	*Protocol*	The Last protocol of IP packet.
[in]	*IsOutbound*	Traffic direction.
[out]	*Action*	The support action of SPD entry.

Return values:

EFI_SUCCESS	Find the related SPD.
EFI_NOT_FOUND	Not find the related SPD entry;

References IpSecMatchIpAddress(), and IpSecMatchNextLayerProtocol().

Referenced by IpSecProcess().

```
BOOLEAN IpSecMatchIpAddress ( IN UINT8                    IpVersion,
                              IN EFI_IP_ADDRESS *         IpAddr,
                              IN EFI_IP_ADDRESS_INFO *    AddressInfo,
                              IN UINT32                   AddressCount
                            )
```

Check if the IP Address in the address range of AddressInfos specified.

Parameters:

[in]	*IpVersion*	The IP version.
[in]	*IpAddr*	Points to EFI_IP_ADDRESS to be check.
[in]	*AddressInfo*	A list of EFI_IP_ADDRESS_INFO that is used to check the IP Address is matched.
[in]	*AddressCount*	The total numbers of the AddressInfo.

Return values:

TRUE	If the Specified IP Address is in the range of the AddressInfos specified.
FALSE	If the Specified IP Address is not in the range of the AddressInfos specified.

Checks if the IP Address in the address range of AddressInfos specified.

Parameters:

[in]	*IpVersion*	The IP version.
[in]	*IpAddr*	Point to EFI_IP_ADDRESS to be check.
[in]	*AddressInfo*	A list of EFI_IP_ADDRESS_INFO that is used to check the IP Address is matched.
[in]	*AddressCount*	The total numbers of the AddressInfo.

Return values:

TRUE	If the Specified IP Address is in the range of the AddressInfos specified.
FALSE	If the Specified IP Address is not in the range of the AddressInfos specified.

References IpSecExtractAddressRange(), and IpSecValidAddressRange().

Referenced by Ikev2MatchSpdEntry(), IpSecLookupPadEntry(), IpSecLookupSadBySpd(), IpSecLookupSadBySpi(), and IpSecLookupSpdEntry().

```
EFI_STATUS EFIAPI IpSecProcess ( IN EFI_IPSEC2_PROTOCOL *                 This,
                                 IN EFI_HANDLE                            NicHandle,
                                 IN UINT8                                 IpVersion,
                                 IN OUT VOID *                            IpHead,
                                 IN OUT UINT8 *                           LastHead,
                                 IN OUT VOID **                           OptionsBuffer,
                                 IN OUT UINT32 *                          OptionsLength,
                                 IN OUT EFI_IPSEC_FRAGMENT_DATA **        FragmentTable,
                                 IN OUT UINT32 *                          FragmentCount,
                                 IN EFI_IPSEC_TRAFFIC_DIR                 TrafficDirection,
```

```
                                    OUT EFI_EVENT *                    RecycleSignal
                                )
```

Handles IPsec packet processing for inbound and outbound IP packets.

The EFI_IPSEC_PROCESS process routine handles each inbound or outbound packet. The behavior is that it can perform one of the following actions: bypass the packet, discard the packet, or protect the packet.

Parameters:

[in]	*This*	Pointer to the EFI_IPSEC2_PROTOCOL instance.
[in]	*NicHandle*	Instance of the network interface.
[in]	*IpVersion*	IPV4 or IPV6.
[in,out]	*IpHead*	Pointer to the IP Header.
[in,out]	*LastHead*	The protocol of the next layer to be processed by IPsec.
[in,out]	*OptionsBuffer*	Pointer to the options buffer.
[in,out]	*OptionsLength*	Length of the options buffer.
[in,out]	*FragmentTable*	Pointer to a list of fragments.
[in,out]	*FragmentCount*	Number of fragments.
[in]	*TrafficDirection*	Traffic direction.
[out]	*RecycleSignal*	Event for recycling of resources.

Return values:

EFI_SUCCESS	The packet was bypassed and all buffers remain the same.
EFI_SUCCESS	The packet was protected.
EFI_ACCESS_DENIED	The packet was discarded.

References IPSEC_PRIVATE_DATA_FROM_IPSEC, IPSEC_SPD_ENTRY_FROM_LIST, IpSecLookupSadEntry(), IpSecLookupSpdEntry(), IpSecProtectInboundPacket(), IpSecProtectOutboundPacket(), IsSubSpdSelector(), mConfigData, and _IPSEC_SPD_ENTRY::Selector.

```
EFI_STATUS IpSecProtectInboundPacket (   IN UINT8                          IpVersion,
                                         IN OUT VOID *                     IpHead,
                                         IN OUT UINT8 *                    LastHead,
                                         IN OUT VOID **                    OptionsBuffer,
                                         IN OUT UINT32 *                   OptionsLength,
                                         IN OUT
                                         EFI_IPSEC_FRAGMENT_DATA           FragmentTable,
                                         **
                                         IN OUT UINT32 *                   FragmentCount,
                                         OUT EFI_IPSEC_SPD_SELECTOR        SpdEntry,
                                         **
                                         OUT EFI_EVENT *                   RecycleEvent
                                     )
```

This function processes the inbound traffic with IPsec.

It checks the received packet security property, trims the ESP/AH header, and then returns without an IPsec protected IP Header and FragmentTable.

Parameters:

[in]	*IpVersion*	The version of IP.

[in,out]	*IpHead*	Points to IP header containing the ESP/AH header to be trimed on input, and without ESP/AH header on return.
[in,out]	*LastHead*	The Last Header in IP header on return.
[in,out]	*OptionsBuffer*	Pointer to the options buffer.
[in,out]	*OptionsLength*	Length of the options buffer.
[in,out]	*FragmentTable*	Pointer to a list of fragments in form of IPsec protected on input, and without IPsec protected on return.
[in,out]	*FragmentCount*	The number of fragments.
[out]	*SpdEntry*	Pointer to contain the address of SPD entry on return.
[out]	*RecycleEvent*	The event for recycling of resources.

Return values:

EFI_SUCCESS	The operation was successful.
EFI_UNSUPPORTED	The IPSEC protocol is not supported.

This function processes the inbound traffic with IPsec.

It checks the received packet security property, trims the ESP/AH header, and then returns without an IPsec protected IP Header and FragmentTable.

Parameters:

[in]	*IpVersion*	The version of IP.
[in,out]	*IpHead*	Points to IP header containing the ESP/AH header to be trimed on input, and without ESP/AH header on return.
[in,out]	*LastHead*	The Last Header in IP header on return.
[in,out]	*OptionsBuffer*	Pointer to the options buffer.
[in,out]	*OptionsLength*	Length of the options buffer.
[in,out]	*FragmentTable*	Pointer to a list of fragments in form of IPsec protected on input, and without IPsec protected on return.
[in,out]	*FragmentCount*	The number of fragments.
[out]	*SpdEntry*	Pointer to contain the address of SPD entry on return.
[out]	*RecycleEvent*	The event for recycling of resources.

Return values:

EFI_SUCCESS	The operation was successful.
EFI_UNSUPPORTED	The IPSEC protocol is not supported.

References IPSEC_ESP_PROTOCOL, and IpSecEspInboundPacket().

Referenced by IpSecProcess().

```
EFI_STATUS
IpSecProtectOutboundPacket ( IN UINT8                          IpVersion,
                             IN OUT VOID *                     IpHead,
                             IN OUT UINT8 *                    LastHead,
                             IN OUT VOID **                    OptionsBuffer,
                             IN OUT UINT32 *                   OptionsLength,
                             IN OUT EFI_IPSEC_FRAGMENT_DATA ** FragmentTable,
                             IN OUT UINT32 *                   FragmentCount,
```

```
                                    IN IPSEC_SAD_ENTRY *          SadEntry,
                                    OUT EFI_EVENT *               RecycleEvent
                                  )
```

This fucntion processes the output traffic with IPsec.

It protected the sending packet by encrypting it payload and inserting ESP/AH header in the orginal IP header, then return the IpHeader and IPsec protected Fragmentable.

Parameters:

[in]	*IpVersion*	The version of IP.
[in,out]	*IpHead*	Point to IP header containing the orginal IP header to be processed on input, and inserted ESP/AH header on return.
[in,out]	*LastHead*	The Last Header in IP header.
[in,out]	*OptionsBuffer*	Pointer to the options buffer.
[in,out]	*OptionsLength*	Length of the options buffer.
[in,out]	*FragmentTable*	Pointer to a list of fragments to be protected by IPsec on input, and with IPsec protected on return.
[in,out]	*FragmentCount*	Number of fragments.
[in]	*SadEntry*	Related SAD entry.
[out]	*RecycleEvent*	Event for recycling of resources.

Return values:

EFI_SUCCESS	The operation is successful.
EFI_UNSUPPORTED	If the IPSEC protocol is not supported.

References IpSecEspOutboundPacket().

Referenced by IpSecProcess().

Variable Documentation

EFI_COMPONENT_NAME_PROTOCOL gIpSecComponentName

Referenced by IpSecComponentNameGetControllerName(), IpSecComponentNameGetDriverName(), and IpSecDriverEntryPoint().

EFI_COMPONENT_NAME2_PROTOCOL gIpSecComponentName2

Referenced by IpSecDriverEntryPoint().

EFI_DPC_PROTOCOL* mDpc

Referenced by IpSecDriverEntryPoint().

EFI_IPSEC2_PROTOCOL mIpSecInstance

Referenced by IpSecDriverEntryPoint().

Generated on Mon Sep 28 08:49:55 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/IpSecMain.c File Reference

Functions

EFI_STATUS EFIAPI IpSecProcess (IN EFI_IPSEC2_PROTOCOL *This, IN EFI_HANDLE NicHandle, IN UINT8 IpVersion, IN OUT VOID *IpHead, IN OUT UINT8 *LastHead, IN OUT VOID **OptionsBuffer, IN OUT UINT32 *OptionsLength, IN OUT EFI_IPSEC_FRAGMENT_DATA **FragmentTable, IN OUT UINT32 *FragmentCount, IN EFI_IPSEC_TRAFFIC_DIR TrafficDirection, OUT EFI_EVENT *RecycleSignal)

Variables

EFI_IPSEC2_PROTOCOL mIpSecInstance = { IpSecProcess, NULL, TRUE }

Detailed Description

The mian interface of IPsec Protocol.

Copyright (c) 2009 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI IpSecProcess ( IN EFI_IPSEC2_PROTOCOL *                This,
                                 IN EFI_HANDLE                           NicHandle,
                                 IN UINT8                                IpVersion,
                                 IN OUT VOID *                           IpHead,
                                 IN OUT UINT8 *                          LastHead,
                                 IN OUT VOID **                          OptionsBuffer,
                                 IN OUT UINT32 *                         OptionsLength,
                                 IN OUT EFI_IPSEC_FRAGMENT_DATA **       FragmentTable,
                                 IN OUT UINT32 *                         FragmentCount,
                                 IN EFI_IPSEC_TRAFFIC_DIR                TrafficDirection,
                                 OUT EFI_EVENT *                         RecycleSignal
                               )
```

Handles IPsec packet processing for inbound and outbound IP packets.

The EFI_IPSEC_PROCESS process routine handles each inbound or outbound packet. The behavior is that it can perform one of the following actions: bypass the packet, discard the packet, or protect the packet.

Parameters:

[in]	*This*	Pointer to the EFI_IPSEC2_PROTOCOL instance.
[in]	*NicHandle*	Instance of the network interface.
[in]	*IpVersion*	IPV4 or IPV6.
[in,out]	*IpHead*	Pointer to the IP Header.
[in,out]	*LastHead*	The protocol of the next layer to be processed by IPsec.
[in,out]	*OptionsBuffer*	Pointer to the options buffer.
[in,out]	*OptionsLength*	Length of the options buffer.
[in,out]	*FragmentTable*	Pointer to a list of fragments.
[in,out]	*FragmentCount*	Number of fragments.
[in]	*TrafficDirection*	Traffic direction.
[out]	*RecycleSignal*	Event for recycling of resources.

Return values:

EFI_SUCCESS	The packet was bypassed and all buffers remain the same.
EFI_SUCCESS	The packet was protected.
EFI_ACCESS_DENIED	The packet was discarded.

References IPSEC_PRIVATE_DATA_FROM_IPSEC, IPSEC_SPD_ENTRY_FROM_LIST, IpSecLookupSadEntry(), IpSecLookupSpdEntry(), IpSecProtectInboundPacket(), IpSecProtectOutboundPacket(), IsSubSpdSelector(), mConfigData, and _IPSEC_SPD_ENTRY::Selector.

Variable Documentation

EFI_IPSEC2_PROTOCOL mIpSecInstance = { IpSecProcess, NULL, TRUE }

Referenced by IpSecDriverEntryPoint().

NetworkPkg/IpSecDxe/Ikev2/ChildSa.c File Reference

Functions

IKE_PACKET * Ikev2CreateChildGenerator (IN UINT8 *SaSession, IN VOID *Context)
EFI_STATUS Ikev2CreateChildParser (IN UINT8 *SaSession, IN IKE_PACKET *IkePacket)
VOID Ikev2ChildSaBeforeDecodePayload (IN UINT8 *SessionCommon, IN UINT8 *PayloadBuf, IN UINTN PayloadSize, IN UINT8 PayloadType)
VOID Ikev2ChildSaAfterEncodePayload (IN UINT8 *SessionCommon, IN UINT8 *PayloadBuf, IN UINTN PayloadSize, IN UINT8 PayloadType)

Variables

IKEV2_PACKET_HANDLER mIkev2CreateChild

Detailed Description

The operations for Child SA.

Copyright (c) 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
VOID Ikev2ChildSaAfterEncodePayload ( IN UINT8 *   SessionCommon,
                                      IN UINT8 *   PayloadBuf,
                                      IN UINTN     PayloadSize,
                                      IN UINT8     PayloadType
                                    )
```

Routine Process after the payload encoding.

Parameters:

[in]	*SessionCommon*	Pointer to ChildSa SessionCommon.
[in]	*PayloadBuf*	Pointer to the payload.
[in]	*PayloadSize*	Size of PayloadBuf in byte.
[in]	*PayloadType*	Type of Payload.

Referenced by Ikev2ChildSaSessionAlloc().

```
VOID Ikev2ChildSaBeforeDecodePayload ( IN UINT8 *   SessionCommon,
                                       IN UINT8 *   PayloadBuf,
                                       IN UINTN     PayloadSize,
                                       IN UINT8     PayloadType
                                     )
```

Routine process before the payload decoding.

Parameters:

[in]	*SessionCommon*	Pointer to ChildSa SessionCommon.
[in]	*PayloadBuf*	Pointer to the payload.
[in]	*PayloadSize*	Size of PayloadBuf in byte.
[in]	*PayloadType*	Type of Payload.

Referenced by Ikev2ChildSaSessionAlloc().

```
IKE_PACKET* Ikev2CreateChildGenerator ( IN UINT8 *   SaSession,
                                        IN VOID *    Context
                                      )
```

Generate IKE Packet for CREATE_CHILD_SA exchange.

This IKE Packet would be the packet for creating new CHILD SA, or the packet for rekeying existing IKE SA, or the packet for existing CHILD SA.

Parameters:

[in]	*SaSession*	Pointer to related SA session.
[in]	*Context*	The data passed by the caller.

return a pointer of IKE packet.

References IKE_HEADER::ExchangeType, IKE_HEADER::Flags, IKE_PACKET::Header, IKE_HEADER_FLAGS_CHILD_INIT, IKE_HEADER_FLAGS_RESPOND, IKE_PACKET_APPEND_PAYLOAD, IKE_XCG_TYPE_CREATE_CHILD_SA, IkePacketAlloc(), IKEV2_CHILD_SA_SESSION::IkeSaSession, IKEV2_SESSION_COMMON::IkeSessionType, IkeSessionTypeChildSa, IKEV2_NOTIFICATION_NO_ADDITIONAL_SAS, IKEV2_PAYLOAD_TYPE_NONE, IKEV2_PAYLOAD_TYPE_NOTIFY, Ikev2GenerateNotifyPayload(), IKEV2_SA_SESSION::InitiatorCookie, IKE_HEADER::InitiatorCookie, IKEV2_SESSION_COMMON::IsInitiator, IKEV2_SA_SESSION::MessageId, IKEV2_CHILD_SA_SESSION::MessageId, IKE_HEADER::MessageId, IKE_HEADER::NextPayload, IKEV2_SA_SESSION::ResponderCookie, IKE_HEADER::ResponderCookie, IKEV2_SA_SESSION::SessionCommon, IKEV2_CHILD_SA_SESSION::SessionCommon, and IKE_HEADER::Version.

```
EFI_STATUS Ikev2CreateChildParser ( IN UINT8 *          SaSession,
                                    IN IKE_PACKET *     IkePacket
                                  )
```

Parse the IKE packet of CREATE_CHILD_SA exchange.

This function parse the IKE packet and save the related information to further calculation.

Parameters:

[in] *SaSession* Pointer to IKEv2_CHILD_SA_SESSION related to this Exchange.
[in] *IkePacket* Received packet to be parsed.

Return values:

EFI_SUCCESS The IKE Packet is acceptable.
EFI_UNSUPPORTED Not support the CREATE_CHILD_SA request.

Variable Documentation

IKEV2_PACKET_HANDLER mIkev2CreateChild

Initial value:

```
{



  Ikev2CreateChildParser,
  Ikev2CreateChildGenerator
}
```

Referenced by Ikev2HandleChildSa(), and Ikev2NegotiateChildSa().

NetworkPkg/IpSecDxe/Ikev2/Exchange.c File Reference

Functions

EFI_STATUS Ikev2NegotiateSa (IN IKE_UDP_SERVICE *UdpService, IN IPSEC_SPD_ENTRY *SpdEntry, IN IPSEC_PAD_ENTRY *PadEntry, IN EFI_IP_ADDRESS *RemoteIp)
EFI_STATUS Ikev2NegotiateChildSa (IN UINT8 *IkeSaSession, IN IPSEC_SPD_ENTRY *SpdEntry, IN UINT8 *Context)
EFI_STATUS Ikev2NegotiateInfo (IN UINT8 *IkeSaSession, IN UINT8 *Context)
VOID Ikev2HandleSa (IN IKE_UDP_SERVICE *UdpService, IN IKE_PACKET *IkePacket)
VOID Ikev2HandleChildSa (IN IKE_UDP_SERVICE *UdpService, IN IKE_PACKET *IkePacket)
VOID Ikev2HandleInfo (IN IKE_UDP_SERVICE *UdpService, IN IKE_PACKET *IkePacket)

Variables

IKE_EXCHANGE_INTERFACE mIkev1Exchange
IKE_EXCHANGE_INTERFACE mIkev2Exchange

Detailed Description

The general interfaces of the IKEv2.

Copyright (c) 2010 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
VOID Ikev2HandleChildSa ( IN IKE_UDP_SERVICE *   UdpService,
                          IN IKE_PACKET *        IkePacket
                        )
```

The general interface when received a IKEv2 packet for the IKE Child SA establishing or IKE SA/CHILD SA rekeying.

This function first find the related IKE SA Session according to the IKE packet's remote IP. Then call the corresponding function to handle this IKE packet according to the related IKE Child Session's State.

Parameters:

[in] *UdpService* Pointer of related UDP Service.
[in] *IkePacket* Data passed by caller.

References IKEV2_PACKET_HANDLER::Generator, IKE_HEADER_FLAGS_RESPOND, IkePacketFree(), IkeRequestTypeCreateChildSa, IkeRequestTypeRekeyChildSa, IkeRequestTypeRekeyIkeSa, IkeSessionTypeIkeSa, Ikev2ChildExchangeRequestType(), Ikev2DecodePacket(), _IPSEC_PRIVATE_DATA::Ikev2EstablishedList, Ikev2SaSessionLookup(), Ikev2SendIkePacket(), Ikev2ValidateHeader(), IPSEC_PRIVATE_DATA_FROM_UDP4LIST, IPSEC_PRIVATE_DATA_FROM_UDP6LIST, IKEV2_SA_SESSION::MessageId, mIkev2CreateChild, IKEV2_PACKET_HANDLER::Parser, and IKEV2_SA_SESSION::SessionCommon.

```
VOID Ikev2HandleInfo ( IN IKE_UDP_SERVICE *   UdpService,
                       IN IKE_PACKET *        IkePacket
                     )
```

It is general interface to handle IKEv2 information Exchange.

Parameters:

[in] *UdpService* Point to IKE UPD Service related to this information exchange.
[in] *IkePacket* The IKE packet to be parsed.

References IkeSessionTypeIkeSa, Ikev2DecodePacket(), _IPSEC_PRIVATE_DATA::Ikev2EstablishedList, Ikev2SaSessionLookup(), Ikev2ValidateHeader(), IPSEC_PRIVATE_DATA_FROM_UDP4LIST, IPSEC_PRIVATE_DATA_FROM_UDP6LIST, mIkev2Info, IKEV2_PACKET_HANDLER::Parser, and IKEV2_SA_SESSION::SessionCommon.

```
VOID Ikev2HandleSa ( IN IKE_UDP_SERVICE *   UdpService,
                     IN IKE_PACKET *        IkePacket
                   )
```

The general interface when received a IKEv2 packet for the IKE SA establishing.

This function first find the related IKE SA Session according to the IKE packet's remote IP. Then call the corresponding function to handle this IKE packet according to the related IKE SA Session's State.

Parameters:

[in] *UdpService* Pointer of related UDP Service.
[in] *IkePacket* Data passed by caller.

References IKEV2_CHILD_SA_SESSION::ByIkeSa, IKEV2_SA_SESSION::ChildSaEstablishSessionList, IKEV2_SA_SESSION::ChildSaSessionList, _IPSEC_PAD_ENTRY::Data, IKEV2_PACKET_HANDLER::Generator, IkeSessionTypeIkeSa, IkeStateAuth, IkeStateIkeSaEstablished, IkeStateInit, IKEV2_CHILD_SA_SESSION_BY_IKE_SA, IKEV2_DUMP_STATE, IKEV2_ESTABLISHING_CHILDSA_LIST, Ikev2ChildSaSessionCreate(), Ikev2ChildSaSessionFree(), Ikev2ChildSaSessionReg(), Ikev2ChildSaSessionRemove(), Ikev2DecodePacket(), Ikev2InitializeSaData(), Ikev2SaSessionAlloc(), Ikev2SaSessionFree(), Ikev2SaSessionInsert(), Ikev2SaSessionLookup(), Ikev2SaSessionReg(), Ikev2SaSessionRemove(), Ikev2SendIkePacket(), _IPSEC_PRIVATE_DATA::Ikev2SessionList, Ikev2ValidateHeader(), IPSEC_PRIVATE_DATA_FROM_UDP4LIST, IPSEC_PRIVATE_DATA_FROM_UDP6LIST, IpSecLookupPadEntry(), IKEV2_SESSION_COMMON::IsInitiator, IKEV2_SESSION_COMMON::LocalPeerIp, IKEV2_CHILD_SA_SESSION::LocalPeerSpi, mIkev2Initial, IKEV2_SA_SESSION::Pad, IKEV2_PACKET_HANDLER::Parser,

IKEV2_SESSION_COMMON::RemotePeerIp, IKEV2_CHILD_SA_SESSION::SaData, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_SA_SESSION::SessionCommon, and IKEV2_SESSION_COMMON::State.

```
EFI_STATUS Ikev2NegotiateChildSa ( IN UINT8 *              IkeSaSession,
                                   IN IPSEC_SPD_ENTRY *    SpdEntry,
                                   IN UINT8 *              Context
                                 )
```

It is general interface to negotiate the Child SA.

There are three situations which will invoke this function. First, create a CHILD SA if the input Context is NULL. Second, rekeying the existing IKE SA if the Context is a IKEv2_SA_SESSION. Third, rekeying the existing CHILD SA if the context is a IKEv2_CHILD_SA_SESSION.

Parameters:

[in]	*IkeSaSession*	Pointer to IKEv2_SA_SESSION related to this operation.
[in]	*SpdEntry*	Pointer to IPSEC_SPD_ENTRY related to this operation.
[in]	*Context*	The data pass from the caller.

Return values:

EFI_SUCCESS	The operation is successful.
EFI_OUT_OF_RESOURCES	The required system resource can't be allocated.
EFI_UNSUPPORTED	The condition is not support yet.

Returns:

Others The operation is failed.

References IKEV2_SA_SESSION::ChildSaSessionList, EFI_IPSEC_ANY_PORT, EFI_IPSEC_ANY_PROTOCOL, IKEV2_PACKET_HANDLER::Generator, IkePacketFree(), IkeStateCreateChild, IKEV2_DUMP_STATE, Ikev2ChildSaSessionAlloc(), Ikev2ChildSaSessionFree(), Ikev2ChildSaSessionInsert(), Ikev2InitializeSaData(), Ikev2SendIkePacket(), IKEV2_SESSION_COMMON::IsInitiator, IKEV2_CHILD_SA_SESSION::LocalPort, mIkev2CreateChild, IKEV2_CHILD_SA_SESSION::ProtoId, IKEV2_CHILD_SA_SESSION::RemotePort, IKEV2_CHILD_SA_SESSION::SaData, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_SA_SESSION::SessionCommon, IKEV2_CHILD_SA_SESSION::Spd, IKEV2_SESSION_COMMON::State, IKEV2_SESSION_COMMON::TimeoutEvent, and IKEV2_SESSION_COMMON::UdpService.

```
EFI_STATUS Ikev2NegotiateInfo ( IN UINT8 *   IkeSaSession,
                                IN UINT8 *   Context
                              )
```

It is general interface to start the Information Exchange.

There are three situations which will invoke this function. First, deliver a Delete Information to delete the IKE SA if the input Context is NULL and the state of related IkeSaSeesion's is on deleting.Second, deliver a Notify Information without the contents if the input Context is NULL. Third, deliver a Notify Information if the input Context is not NULL.

Parameters:

[in]	*IkeSaSession*	Pointer to IKEv2_SA_SESSION related to this operation.

[in] *Context* Data passed by caller.

Return values:

EFI_SUCCESS	The operation is successful.
EFI_OUT_OF_RESOURCES	The required system resource can't be allocated.
EFI_UNSUPPORTED	The condition is not support yet.

Returns:

Otherwise The operation is failed.

References IKEV2_SA_SESSION::DeleteSaList, IKEV2_PACKET_HANDLER::Generator, IkePacketFree(), IkeStateSaDeleting, IKEV2_CHILD_SA_SESSION_BY_DEL_SA, Ikev2SendIkePacket(), mIkev2Info, IKE_UDP_SERVICE::Output, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_SA_SESSION::SessionCommon, IKEV2_SESSION_COMMON::State, and IKEV2_SESSION_COMMON::UdpService.

```
EFI_STATUS Ikev2NegotiateSa ( IN IKE_UDP_SERVICE *    UdpService,
                              IN IPSEC_SPD_ENTRY *    SpdEntry,
                              IN IPSEC_PAD_ENTRY *    PadEntry,
                              IN EFI_IP_ADDRESS *     RemoteIp
                            )
```

General interface to intialize a IKEv2 negotiation.

Parameters:

[in]	*UdpService*	Point to Udp Servcie used for the IKE packet sending.
[in]	*SpdEntry*	Point to SPD entry related to this IKE negotiation.
[in]	*PadEntry*	Point to PAD entry related to this IKE negotiation.
[in]	*RemoteIp*	Point to IP Address which the remote peer to negnotiate.

Return values:

EFI_SUCCESS	The operation is successful.
EFI_OUT_OF_RESOURCES	The required system resource can't be allocated.
EFI_INVALID_PARAMETER	If UdpService or RemoteIp is NULL.

Returns:

Others The operation is failed.

References _IPSEC_PAD_ENTRY::Data, IKEV2_PACKET_HANDLER::Generator, IkePacketFree(), IkeStateInit, IKEV2_DUMP_STATE, IKEV2_TRANSFORM_ID_DH_1024MODP, Ikev2InitializeSaData(), Ikev2SaSessionAlloc(), Ikev2SaSessionFree(), Ikev2SaSessionInsert(), Ikev2SaSessionLookup(), Ikev2SendIkePacket(), _IPSEC_PRIVATE_DATA::Ikev2SessionList, IPSEC_PRIVATE_DATA_FROM_UDP4LIST, IPSEC_PRIVATE_DATA_FROM_UDP6LIST, IKEV2_SESSION_COMMON::IsInitiator, IKEV2_SESSION_COMMON::LocalPeerIp, mIkev2Initial, IKEV2_SA_SESSION::Pad, IKEV2_SESSION_COMMON::PreferDhGroup, IKEV2_SESSION_COMMON::RemotePeerIp, IKEV2_SA_SESSION::SaData, IKEV2_SA_SESSION::SessionCommon, IKEV2_SA_SESSION::Spd, and IKEV2_SESSION_COMMON::State.

Variable Documentation

IKE_EXCHANGE_INTERFACE mIkev1Exchange

Initial value:

```
{
 1,
 NULL,
 NULL,
 NULL,
 NULL,
 NULL,
 NULL,
}
```

IKE_EXCHANGE_INTERFACE mIkev2Exchange

Initial value:

```
{
 2,
 Ikev2NegotiateSa,
 Ikev2NegotiateChildSa,
 Ikev2NegotiateInfo,
 Ikev2HandleSa,
 Ikev2HandleChildSa,
 Ikev2HandleInfo
}
```

Referenced by Ikev2LifetimeNotify().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/Ikev2/Ikev2.h File Reference

Data Structures

struct IKEV2_DH_BUFFER
struct IKEV2_SESSION_KEYS
struct IKEV2_SA_PARAMS
struct IKEV2_SA_DATA
struct IKEV2_PROPOSAL_DATA
struct IKEV2_TRANSFORM_DATA
struct IKEV2_SESSION_COMMON
struct IKEV2_SA_SESSION
struct IKEV2_CHILD_SA_SESSION
struct IKEV2_INFO_EXCHANGE_CONTEXT
struct PRF_DATA_FRAGMENT
struct IKEV2_PACKET_HANDLER

Defines

"memItemRight" valign="bottom">IKEV2_TS_ANY_PORT 0xffff

#define IKEV2_TS_ANY_PROTOCOL 0
#define IKEV2_DELET_CHILDSA_LIST 0
#define IKEV2_ESTABLISHING_CHILDSA_LIST 1
#define IKEV2_ESTABLISHED_CHILDSA_LIST 2
#define IKEV2_SA_SESSION_SIGNATURE SIGNAT
('I', 'K', 'E', 'I')
#define IKEV2_SA_SESSION_FROM_COMMON(a)
IKEV2_SA_SESSION, SessionCommon,
IKEV2_SA_SESSION_SIGNATURE)
#define IKEV2_SA_SESSION_BY_SESSION(a) CR (
IKEV2_SA_SESSION, BySessionTable,
IKEV2_SA_SESSION_SIGNATURE)
#define IKEV2_SA_SESSION_BY_ESTABLISHED(a)
(a, IKEV2_SA_SESSION, ByEstablishedTable,
IKEV2_SA_SESSION_SIGNATURE)
#define IKEV2_CHILD_SA_SESSION_SIGNATURE
('I', 'K', 'E', 'C')
#define IKEV2_CHILD_SA_SESSION_FROM_COMM
(a, IKEV2_CHILD_SA_SESSION, SessionCom
IKEV2_CHILD_SA_SESSION_SIGNATURE)
#define IKEV2_CHILD_SA_SESSION_BY_IKE_SA(a
(a, IKEV2_CHILD_SA_SESSION, ByIkeSa,
IKEV2_CHILD_SA_SESSION_SIGNATURE)
#define IKEV2_CHILD_SA_SESSION_BY_DEL_SA(a
(a, IKEV2_CHILD_SA_SESSION, ByDelete,
IKEV2_CHILD_SA_SESSION_SIGNATURE)
#define IS_IKEV2_SA_SESSION(s) ((s)->Common.Ik
== IkeSessionTypeIkeSa)
#define IKEV2_SA_FIRST_PROPOSAL(Sa) (IKEV2_
*)((IKEV2_SA *)(Sa)+1)
#define IKEV2_NEXT_TRANSFORM_WITH_SIZE(T
TransformSize) (IKEV2_TRANSFORM *) ((U

*)(Transform) + (TransformSize))
#define IKEV2_NEXT_PROPOSAL_WITH_SIZE(Proposal, ProposalSize) (IKEV2_PROPOSAL *) ((UINT8 *)(Proposal) + (ProposalSize))
#define IKEV2_PROPOSAL_FIRST_TRANSFORM(Proposal)
#define IKEV2_PROPOSAL_FIRST_TRANSFORM(Proposal)

Typedefs

typedef IKE_PACKET *(* IKEV2_PACKET_GENERATOR)(IN UINT8 *SaSession, IN VOID *Context)
typedef EFI_STATUS(* IKEV2_PACKET_PARSER)(IN UINT8 *SaSession, IN IKE_PACKET *IkePacket)

Enumerations

enum IKEV2_SESSION_STATE { IkeStateInit, IkeStateAuth, IkeStateIkeSaEstablished, IkeStateCreateChild, IkeStateSaRekeying, IkeStateChildSaEstablished, IkeStateSaDeleting, IkeStateMaximum }
enum IKEV2_CREATE_CHILD_REQUEST_TYPE { IkeRequestTypeCreateChildSa, IkeRequestTypeRekeyChildSa, IkeRequestTypeRekeyIkeSa, IkeRequestTypeMaximum }
enum IKEV2_INFO_TYPE { Ikev2InfoNotify, Ikev2InfoDelete, Ikev2InfoLiveCheck }

Variables

IKEV2_PACKET_HANDLER mIkev2Initial [][2]
IKEV2_PACKET_HANDLER mIkev2CreateChild
IKEV2_PACKET_HANDLER mIkev2Info

Detailed Description

IKEv2 related definitions.

Copyright (c) 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"paramtype">a) CR (a, IKEV2_CHILD_SA_SESSION, ByDelete, IKEV2_CHILD_SA_SESSION_SIGNATURE)

Referenced by Ikev2ChildSaSessionRemove(), and Ikev2NegotiateInfo().

#define IKEV2_CHILD_SA_SESSION_BY_IKE_SA (a) CR (a, IKEV2_CHILD_SA_SESSION, ByIkeSa, IKEV2_CHILD_SA_SESSION_SIGNATURE)

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), Ikev2ChildSaSessionLookupByMid(), Ikev2ChildSaSessionLookupBySpi(), Ikev2ChildSaSessionRemove(), Ikev2HandleSa(), Ikev2InfoParser(), Ikev2ResendNotify(), and Ikev2SaSessionFree().

#define IKEV2_CHILD_SA_SESSION_FROM_COMMON (a) CR (a, IKEV2_CHILD_SA_SESSION, SessionCommon, IKEV2_CHILD_SA_SESSION_SIGNATURE)

Referenced by Ikev2DecryptPacket(), Ikev2EncryptPacket(), Ikev2InitializeSaData(), Ikev2LifetimeNotify(), and Ikev2ResendNotify().

#define IKEV2_CHILD_SA_SESSION_SIGNATURE SIGNATURE_32 ('I', 'K', 'E', 'C')

Referenced by Ikev2ChildSaSessionAlloc().

#define IKEV2_DELET_CHILDSA_LIST 0

Referenced by Ikev2ChildSaSessionRemove(), Ikev2OnPacketSent(), and Ikev2ResendNotify().

#define IKEV2_ESTABLISHED_CHILDSA_LIST 2

Referenced by Ikev2ChildSaSessionReg(), Ikev2ChildSaSessionRemove(), and Ikev2ChildSaSilentDelete().

#define IKEV2_ESTABLISHING_CHILDSA_LIST 1

Referenced by Ikev2ChildSaSessionRemove(), Ikev2HandleSa(), and Ikev2ResendNotify().

#define IKEV2_NEXT_PROPOSAL_WITH_SIZE (Proposal, ProposalSize) (IKEV2_PROPOSAL *) ((UINT8 *)(Proposal) + (ProposalSize))

Referenced by Ikev2DecodeSa().

#define IKEV2_NEXT_TRANSFORM_WITH_SIZE (Transform, TransformSize) (IKEV2_TRANSFORM *) ((UINT8 *)(Transform) + (TransformSize))

Referenced by Ikev2DecodeSa().

#define IKEV2_PROPOSAL_FIRST_TRANSFORM (Proposal)

Value:

```
(IKEV2_TRANSFORM *)((UINT8 *)((IKEV2_PROPOSAL *)(Proposal)+1) + \
                    (((IKEV2_PROPOSAL *)(Proposal))->SpiSize))
```

#define IKEV2_PROPOSAL_FIRST_TRANSFORM (Proposal)

Value:

```
(IKEV2_TRANSFORM *)((UINT8 *)((IKEV2_PROPOSAL *)(Proposal)+1) + \
                    (((IKEV2_PROPOSAL *)(Proposal))->SpiSize))
```

Referenced by Ikev2DecodeSa().

#define IKEV2_SA_FIRST_PROPOSAL (Sa) (IKEV2_PROPOSAL *)((IKEV2_SA *)(Sa)+1)

Referenced by Ikev2DecodeSa().

#define IKEV2_SA_SESSION_BY_ESTABLISHED (a) CR (a, IKEV2_SA_SESSION, ByEstablishedTable, IKEV2_SA_SESSION_SIGNATURE)

#define IKEV2_SA_SESSION_BY_SESSION (a) CR (a, IKEV2_SA_SESSION, BySessionTable, IKEV2_SA_SESSION_SIGNATURE)

Referenced by IkeDeleteAllSas(), Ikev2SaSessionLookup(), Ikev2SaSessionRemove(), and IpSecStop().

#define IKEV2_SA_SESSION_FROM_COMMON (a) CR (a, IKEV2_SA_SESSION, SessionCommon, IKEV2_SA_SESSION_SIGNATURE)

Referenced by Ikev2DecodePacket(), Ikev2DecryptPacket(), Ikev2EncodePacket(), Ikev2EncryptPacket(), Ikev2LifetimeNotify(), and Ikev2ResendNotify().

#define IKEV2_SA_SESSION_SIGNATURE SIGNATURE_32 ('I', 'K', 'E', 'I')

Referenced by IkeGetVersionFromSession(), and Ikev2SaSessionAlloc().

#define IKEV2_TS_ANY_PORT 0xffff

Referenced by Ikev2GenerateTsPayload().

#define IKEV2_TS_ANY_PROTOCOL 0

Referenced by Ikev2GenerateTsPayload().

#define IS_IKEV2_SA_SESSION (s) ((s)->Common.IkeSessionType == IkeSessionTypeIkeSa)

Typedef Documentation

typedef IKE_PACKET*(* IKEV2_PACKET_GENERATOR)(IN UINT8 *SaSession, IN VOID *Context)

typedef EFI_STATUS(* IKEV2_PACKET_PARSER)(IN UINT8 *SaSession, IN IKE_PACKET *IkePacket)

Enumeration Type Documentation

enum IKEV2_CREATE_CHILD_REQUEST_TYPE

Enumerator:

IkeRequestTypeCreateChildSa
IkeRequestTypeRekeyChildSa
IkeRequestTypeRekeyIkeSa
IkeRequestTypeMaximum

enum IKEV2_INFO_TYPE

Enumerator:

Ikev2InfoNotify
Ikev2InfoDelete
Ikev2InfoLiveCheck

enum IKEV2_SESSION_STATE

Enumerator:

IkeStateInit
IkeStateAuth
IkeStateIkeSaEstablished
IkeStateCreateChild
IkeStateSaRekeying
IkeStateChildSaEstablished
IkeStateSaDeleting
IkeStateMaximum

Variable Documentation

IKEV2_PACKET_HANDLER mIkev2CreateChild

Referenced by Ikev2HandleChildSa(), and Ikev2NegotiateChildSa().

IKEV2_PACKET_HANDLER mIkev2Info

Referenced by Ikev2HandleInfo(), and Ikev2NegotiateInfo().

IKEV2_PACKET_HANDLER mIkev2Initial[][2]

Referenced by Ikev2HandleSa(), and Ikev2NegotiateSa().

NetworkPkg/IpSecDxe/Ikev2/Info.c File Reference

Functions

IKE_PACKET * Ikev2InfoGenerator (IN UINT8 *SaSession, IN VOID *Context)
EFI_STATUS Ikev2InfoParser (IN UINT8 *SaSession, IN IKE_PACKET *IkePacket)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED
IKEV2_PACKET_HANDLER mIkev2Info

Detailed Description

The Implementations for Information Exchange.

(C) Copyright 2015 Hewlett-Packard Development Company, L.P.
Copyright (c) 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

IKE_PACKET* Ikev2InfoGenerator (IN UINT8 * *SaSession*,
IN VOID * *Context*
)

Generate Information Packet.

The information Packet may contain one Delete Payload, or Notify Payload, which dependes on the Context's parameters.

Parameters:

[in]	*SaSession*	Pointer to IKE SA Session or Child SA Session which is related to the information Exchange.
[in]	*Context*	The Data passed from the caller. If the Context is not NULL it should contain the information for Notification Data.

Return values:

Pointer of IKE_PACKET generated.

References IKE_HEADER::ExchangeType, IKE_HEADER::Flags, IKE_PACKET::Header, IKE_HEADER_FLAGS_INIT, IKE_HEADER_FLAGS_RESPOND, IKE_PACKET_APPEND_PAYLOAD, IkePacketAlloc(), IKEV2_CHILD_SA_SESSION::IkeSaSession, IKEV2_SESSION_COMMON::IkeSessionType, IkeSessionTypeIkeSa, IkeStateSaDeleting, IKEV2_EXCHANGE_TYPE_INFO, IKEV2_PAYLOAD_TYPE_DELETE, IKEV2_PAYLOAD_TYPE_NONE, Ikev2GenerateDeletePayload(), Ikev2InfoLiveCheck, Ikev2InfoNotify, Ikev2SaSessionIncreaseMessageId(), IKEV2_INFO_EXCHANGE_CONTEXT::InfoType, IKEV2_SA_SESSION::InitiatorCookie, IKE_HEADER::InitiatorCookie, IKE_PACKET::IsDeleteInfo, IKEV2_SESSION_COMMON::IsInitiator, IKEV2_SESSION_COMMON::IsOnDeleting, IKEV2_CHILD_SA_SESSION::LocalPeerSpi, IKEV2_CHILD_SA_SESSION::MessageId, IKEV2_SA_SESSION::MessageId, IKEV2_INFO_EXCHANGE_CONTEXT::MessageId, IKE_HEADER::MessageId, IKE_HEADER::NextPayload, IKEV2_SESSION_COMMON::Private, IKE_PACKET::Private, IKEV2_SA_SESSION::ResponderCookie, IKE_HEADER::ResponderCookie, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_SA_SESSION::SessionCommon, IKE_PACKET::Spi, IKEV2_SESSION_COMMON::State, and IKE_HEADER::Version.

Referenced by Ikev2InfoParser().

```
EFI_STATUS Ikev2InfoParser ( IN UINT8 *          SaSession,
                             IN IKE_PACKET *     IkePacket
                           )
```

Parse the Info Exchange.

Parameters:

[in] *SaSession* Pointer to IKEV2_SA_SESSION.

[in] *IkePacket* Pointer to IkePacket related to the Information Exchange.

Return values:

EFI_SUCCESS The operation finised successed.

References IKEV2_CHILD_SA_SESSION::ByDelete, IKEV2_SA_SESSION::BySessionTable, IKEV2_SA_SESSION::ChildSaEstablishSessionList, IKEV2_SA_SESSION::DeleteSaList, IKE_HEADER_FLAGS_INIT, IKE_PAYLOAD_BY_PACKET, IkeStateSaDeleting, _IPSEC_PRIVATE_DATA::Ikev1EstablishedList, IKEV2_CHILD_SA_SESSION_BY_IKE_SA, IKEV2_PAYLOAD_TYPE_DELETE, IKEV2_PAYLOAD_TYPE_NONE, Ikev2ChildSaSilentDelete(), _IPSEC_PRIVATE_DATA::Ikev2EstablishedList, Ikev2InfoDelete, Ikev2InfoGenerator(), Ikev2InfoLiveCheck, Ikev2SaSessionFree(), Ikev2SendIkePacket(), IKEV2_INFO_EXCHANGE_CONTEXT::InfoType, _IPSEC_PRIVATE_DATA::IpSec, IPSEC_STATUS_DISABLED, IPSECCONFIG_STATUS_NAME, IKEV2_SESSION_COMMON::IsInitiator, _IPSEC_PRIVATE_DATA::IsIPsecDisabling, IKEV2_CHILD_SA_SESSION::MessageId, IKEV2_INFO_EXCHANGE_CONTEXT::MessageId, IKEV2_DELETE::NumSpis, IKE_PAYLOAD::PayloadBuf, IKE_PAYLOAD::PayloadType, IKEV2_SESSION_COMMON::Private, IKEV2_CHILD_SA_SESSION::RemotePeerSpi, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_SA_SESSION::SessionCommon, IKEV2_DELETE::SpiSize, and IKEV2_SESSION_COMMON::State.

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED IKEV2_PACKET_HANDLER mIkev2Info

Initial value:

```
{
```

```
  Ikev2InfoParser,
  Ikev2InfoGenerator
}
```

Referenced by Ikev2HandleInfo(), and Ikev2NegotiateInfo().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/Ikev2/Payload.c File Reference

Defines

"memItemRight" valign="bottom">CONSTANT_KEY_SIZE 17

Functions

IKE_PAYLOAD * Ikev2GenerateSaPayload (IN IKEV2_SA_DATA *SessionSaData, IN UINT8 NextPayload, IN IKE_SESSION_TYPE Type)

IKE_PAYLOAD * Ikev2GenerateNoncePayload (IN UINT8 *NonceBuf, IN UINTN NonceSize, IN UINT8 NextPayload)

IKE_PAYLOAD * Ikev2GenerateKePayload (IN OUT IKEV2_SA_SESSION *IkeSaSession, IN UINT8 NextPayload)

IKE_PAYLOAD * Ikev2GenerateIdPayload (IN IKEV2_SESSION_COMMON *CommonSession, IN UINT8 NextPayload)

IKE_PAYLOAD * Ikev2GenerateCertIdPayload (IN IKEV2_SESSION_COMMON *CommonSession, IN UINT8 NextPayload, IN UINT8 *InCert, IN UINTN CertSize)

IKE_PAYLOAD * Ikev2PskGenerateAuthPayload (IN IKEV2_SA_SESSION *IkeSaSession, IN IKE_PAYLOAD *IdPayload, IN UINT8 NextPayload, IN BOOLEAN IsVerify)

IKE_PAYLOAD * Ikev2CertGenerateAuthPayload (IN IKEV2_SA_SESSION *IkeSaSession, IN IKE_PAYLOAD *IdPayload, IN UINT8 NextPayload, IN BOOLEAN IsVerify, IN UINT8 *UefiPrivateKey, IN UINTN UefiPrivateKeyLen, IN UINT8 *UefiKeyPwd, IN UINTN UefiKeyPwdLen)

IKE_PAYLOAD * Ikev2GenerateTsPayload (IN IKEV2_CHILD_SA_SESSION *ChildSa, IN UINT8 NextPayload, IN BOOLEAN IsTunnel)

IKE_PAYLOAD * Ikev2GenerateNotifyPayload (IN UINT8 ProtocolId, IN UINT8 NextPayload, IN UINT8 SpiSize, IN UINT16 MessageType, IN UINT8 *SpiBuf, IN UINT8 *NotifyData, IN UINTN NotifyDataSize)

IKE_PAYLOAD *

	Ikev2GenerateDeletePayload (IN IKEV2_SA_SESSION *IkeSaSession, IN UINT8 NextPayload, IN UINT8 SpiSize, IN UINT16 SpiNum, IN UINT8 *SpiBuf)
IKE_PAYLOAD *	Ikev2GenerateCpPayload (IN IKEV2_SA_SESSION *IkeSaSession, IN UINT8 NextPayload, IN UINT8 CfgType)
EFI_STATUS	Ikev2ParserNotifyCookiePayload (IN IKE_PAYLOAD *IkeNCookie, IN OUT IKEV2_SA_SESSION *IkeSaSession)
IKE_PAYLOAD *	Ikev2GenerateCertificatePayload (IN IKEV2_SA_SESSION *IkeSaSession, IN UINT8 NextPayload, IN UINT8 *Certificate, IN UINTN CertificateLen, IN UINT8 EncodeType, IN BOOLEAN IsRequest)
VOID	ClearAllPayloads (IN IKE_PACKET *IkePacket)
IKEV2_SA *	Ikev2EncodeSa (IN IKEV2_SESSION_COMMON *SessionCommon, IN IKEV2_SA_DATA *SaData)
IKEV2_SA_DATA *	Ikev2DecodeSa (IN IKEV2_SESSION_COMMON *SessionCommon, IN IKEV2_SA *Sa)
EFI_STATUS	Ikev2EncodePayload (IN UINT8 *SessionCommon, IN OUT IKE_PAYLOAD *IkePayload)
EFI_STATUS	Ikev2DecodePayload (IN UINT8 *SessionCommon, IN OUT IKE_PAYLOAD *IkePayload)
EFI_STATUS	Ikev2DecodePacket (IN IKEV2_SESSION_COMMON *SessionCommon, IN OUT IKE_PACKET *IkePacket, IN UINTN IkeType)
EFI_STATUS	Ikev2EncodePacket (IN IKEV2_SESSION_COMMON *SessionCommon, IN OUT IKE_PACKET *IkePacket, IN UINTN IkeType)
EFI_STATUS	Ikev2DecryptPacket (IN IKEV2_SESSION_COMMON *SessionCommon, IN OUT IKE_PACKET *IkePacket, IN OUT UINTN IkeType)
EFI_STATUS	Ikev2EncryptPacket (IN IKEV2_SESSION_COMMON *SessionCommon, IN OUT IKE_PACKET *IkePacket)
VOID	Ikev2OnPacketAccepted (IN IKEV2_SESSION_COMMON *SessionCommon, IN IKE_PACKET *IkePacket, IN UINT8 IkeType)

VOID EFIAPI Ikev2OnPacketSent (IN NET_BUF *Packet, IN UDP_END_POINT *EndPoint, IN EFI_STATUS IoStatus, IN VOID *Context)

EFI_STATUS Ikev2SendIkePacket (IN IKE_UDP_SERVICE *IkeUdpService, IN UINT8 *SessionCommon, IN IKE_PACKET *IkePacket, IN UINTN IkeType)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED CHAR8 mConstantKey [CONSTANT_KEY_SIZE]

Detailed Description

The implementation of Payloads Creation.

(C) Copyright 2015 Hewlett-Packard Development Company, L.P.
Copyright (c) 2010 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Ikev2PskGenerateAuthPayload().

Function Documentation

VOID ClearAllPayloads (IN IKE_PACKET * *IkePacket*)

Remove and free all IkePayloads in the specified IkePacket.

Parameters:

`[in]` *IkePacket* The pointer of IKE_PACKET.

References IKE_PACKET_REMOVE_PAYLOAD, IKE_PAYLOAD_BY_PACKET, and IkePayloadFree().

Referenced by Ikev2DecodePacket(), and Ikev2EncryptPacket().

```
IKE_PAYLOAD* Ikev2CertGenerateAuthPayload ( IN IKEV2_SA_SESSION *   IkeSaSession,
                                            IN IKE_PAYLOAD *        IdPayload,
                                            IN UINT8                NextPayload,
                                            IN BOOLEAN              IsVerify,
                                            IN UINT8 *              UefiPrivateKey,
                                            IN UINTN                UefiPrivateKeyLen,
                                            IN UINT8 *              UefiKeyPwd,
                                            IN UINTN                UefiKeyPwdLen
                                          )
```

Generate a Authentication Payload for Certificate Auth method.

This function has two functions. One is creating a local Authentication Payload for sending and other is creating the remote Authentication data for verification when the IsVerify is TURE.

Parameters:

[in]	*IkeSaSession*	Pointer to IKEV2_SA_SESSION related to.
[in]	*IdPayload*	Pointer to the ID payload to be used for Authentication payload generation.
[in]	*NextPayload*	The type filled into the Authentication Payload next payload field.
[in]	*IsVerify*	If it is TURE, the Authentication payload is used for verification.
[in]	*UefiPrivateKey*	Pointer to the UEFI private key. Ignore it when verify the authenticate payload.
[in]	*UefiPrivateKeyLen*	The size of UefiPrivateKey in bytes. Ignore it when verify the authenticate payload.
[in]	*UefiKeyPwd*	Pointer to the password of UEFI private key. Ignore it when verify the authenticate payload.
[in]	*UefiKeyPwdLen*	The size of UefiKeyPwd in bytes.Ignore it when verify the authenticate payload.

Returns:

pointer to IKE Authentication payload for Cerifitcation method.

References IKEV2_AUTH::AuthMethod, PRF_DATA_FRAGMENT::Data, PRF_DATA_FRAGMENT::DataSize, IKEV2_AUTH::Header, IkePayloadAlloc(), IkePayloadFree(), IKEV2_AUTH_METHOD_RSA, IKEV2_PAYLOAD_TYPE_AUTH, IpSecCryptoIoAuthDataWithCertificate(), IpSecCryptoIoHash(), IpSecCryptoIoHmac(), IpSecDumpBuf(), IpSecGetHmacDigestLength(), IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2AuthCertGenerator(), and Ikev2AuthCertParser().

```
EFI_STATUS Ikev2DecodePacket ( IN IKEV2_SESSION_COMMON *   SessionCommon,
                               IN OUT IKE_PACKET *         IkePacket,
                               IN UINTN                    IkeType
                             )
```

Decode the IKE packet.

This function first decrypts the IKE packet if needed , then separates the whole IKE packet from the IkePacket->PayloadBuf into IkePacket payload list.

Parameters:

[in]	*SessionCommon*	Pointer to IKEV1_SESSION_COMMON containing some parameter used by IKE packet decoding.
[in,out]	*IkePacket*	The IKE Packet to be decoded on input, and the decoded result on return.
[in]	*IkeType*	The type of IKE. IKE_SA_TYPE, IKE_INFO_TYPE and IKE_CHILD_TYPE are supported.

Return values:

EFI_SUCCESS	The IKE packet is decoded successfully.
Otherwise	The IKE packet decoding is failed.

References ClearAllPayloads(), IKE_PACKET_APPEND_PAYLOAD, IkeHdrNetToHost(), IkePayloadAlloc(), IkeStateAuth, IKEV2_EXCHANGE_TYPE_INFO, IKEV2_EXCHANGE_TYPE_INIT, IKEV2_PAYLOAD_TYPE_NONE, IKEV2_SA_SESSION_FROM_COMMON, Ikev2DecodePayload(), Ikev2DecryptPacket(), IKEV2_SA_SESSION::InitPacket, IKEV2_SA_SESSION::InitPacketSize, IPSEC_DUMP_BUF, IKE_PAYLOAD::IsPayloadBufExt, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, IKEV2_SA_SESSION::RespPacket, and IKEV2_SA_SESSION::RespPacketSize.

Referenced by Ikev2HandleChildSa(), Ikev2HandleInfo(), and Ikev2HandleSa().

```
EFI_STATUS Ikev2DecodePayload ( IN UINT8 *            SessionCommon,
                                IN OUT IKE_PAYLOAD *  IkePayload
                              )
```

The general interface for decoding Payload.

This function converts the received Payload into internal structure.

Parameters:

[in]	*SessionCommon*	Pointer to IKE Session Common used for decoding.
[in,out]	*IkePayload*	Pointer to IKE payload to be decoded as input, and store the decoded result as output.

Return values:

EFI_INVALID_PARAMETER	Meet error when decoding the SA payload.
EFI_SUCCESS	Decoded successfully.

References IKEV2_CFG_ATTRIBUTES::AttritType, IKEV2_KEY_EXCHANGE::DhGroup, TRAFFIC_SELECTOR::EndPort, IKEV2_DUMP_PAYLOAD, IKEV2_PAYLOAD_TYPE_AUTH, IKEV2_PAYLOAD_TYPE_CP, IKEV2_PAYLOAD_TYPE_DELETE, IKEV2_PAYLOAD_TYPE_ID_INIT, IKEV2_PAYLOAD_TYPE_ID_RSP, IKEV2_PAYLOAD_TYPE_KE, IKEV2_PAYLOAD_TYPE_NOTIFY, IKEV2_PAYLOAD_TYPE_SA, IKEV2_PAYLOAD_TYPE_TS_INIT, IKEV2_PAYLOAD_TYPE_TS_RSP, Ikev2DecodeSa(), IKEV2_NOTIFY::MessageType, IKEV2_DELETE::NumSpis, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, TRAFFIC_SELECTOR::SelecorLen, TRAFFIC_SELECTOR::StartPort, IKEV2_TS::TSNumbers, and IKEV2_CFG_ATTRIBUTES::ValueLength.

Referenced by Ikev2DecodePacket().

```
IKEV2_SA_DATA* Ikev2DecodeSa ( IN IKEV2_SESSION_COMMON *   SessionCommon,
                               IN IKEV2_SA *               Sa
                             )
```

Decode SA payload.

This function converts the received SA payload to internal data structure.

Parameters:

[in]	*SessionCommon*	Pointer to IKE Common Session used to decode the SA Payload.
[in]	*Sa*	Pointer to SA Payload

Returns:

a Pointer to internal data structure for SA payload.

References IKE_SA_ATTRIBUTE::Attr, IKEV2_TRANSFORM_DATA::Attribute, IKE_SA_ATTRIBUTE::AttrType, IKE_SA_ATTR_UNION::AttrValue, IKEV2_TRANSFORM::Header, IKEV2_PROPOSAL::Header, IKE_PROPOSAL_NEXT_PAYLOAD_MORE, IKEV2_ATTRIBUTE_TYPE_KEYLEN, IKEV2_NEXT_PROPOSAL_WITH_SIZE, IKEV2_NEXT_TRANSFORM_WITH_SIZE, IKEV2_PROPOSAL_FIRST_TRANSFORM, IKEV2_SA_FIRST_PROPOSAL, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKEV2_SA_DATA::NumProposals, IKEV2_PROPOSAL_DATA::NumTransforms, IKEV2_PROPOSAL::NumTransforms, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKEV2_PROPOSAL::ProposalIndex, IKEV2_PROPOSAL_DATA::ProposalIndex, IKEV2_PROPOSAL::ProtocolId, IKEV2_PROPOSAL_DATA::ProtocolId, SA_ATTR_FORMAT_BIT, IKEV2_PROPOSAL_DATA::Spi, IKEV2_PROPOSAL::SpiSize, IKEV2_TRANSFORM::TransformId, IKEV2_TRANSFORM_DATA::TransformId, IKEV2_TRANSFORM::TransformType, and IKEV2_TRANSFORM_DATA::TransformType.

Referenced by Ikev2DecodePayload().

```
EFI_STATUS Ikev2DecryptPacket ( IN IKEV2_SESSION_COMMON *   SessionCommon,
                                IN OUT IKE_PACKET *         IkePacket,
                                IN OUT UINTN                IkeType
                              )
```

Decrypt IKE packet.

This function decrypts the Encrypted IKE packet and put the result into IkePacket->PayloadBuf.

Parameters:

[in]	*SessionCommon*	Pointer to IKEV2_SESSION_COMMON containing some parameter used during decrypting.
[in,out]	*IkePacket*	Pointer to IKE_PACKET to be decrypted as input, and the decrypted result as output.
[in,out]	*IkeType*	The type of IKE. IKE_SA_TYPE, IKE_INFO_TYPE and IKE_CHILD_TYPE are supportted.

Return values:

EFI_INVALID_PARAMETER	If the IKE packet length is zero or the IKE packet length is not aligned with Algorithm Block Size
EFI_SUCCESS	Decrypt IKE packet successfully.

References HASH_DATA_FRAGMENT::Data, HASH_DATA_FRAGMENT::DataSize, IKEV2_SA_PARAMS::EncAlgId, IkeHdrHostToNet(), IKEV2_SA_SESSION::IkeKeys, IKEV2_CHILD_SA_SESSION::IkeSaSession, IkeSessionTypeChildSa, IkeSessionTypeIkeSa, IKEV2_CHILD_SA_SESSION_FROM_COMMON, IKEV2_PAYLOAD_TYPE_ENCRYPT, IKEV2_SA_SESSION_FROM_COMMON, IKEV2_SA_PARAMS::IntegAlgId, IPSEC_DUMP_BUF, IpSecCryptoIoDecrypt(), IpSecCryptoIoHmac(), IpSecGetEncryptBlockSize(), IpSecGetIcvLength(), IKEV2_SESSION_COMMON::SaParams, IKEV2_SA_SESSION::SessionCommon, IKEV2_SESSION_KEYS::SkAiKey, IKEV2_SESSION_KEYS::SkAiKeySize, IKEV2_SESSION_KEYS::SkArKey, IKEV2_SESSION_KEYS::SkArKeySize, IKEV2_SESSION_KEYS::SkEiKey, IKEV2_SESSION_KEYS::SkEiKeySize, IKEV2_SESSION_KEYS::SkErKey, and IKEV2_SESSION_KEYS::SkErKeySize.

Referenced by Ikev2DecodePacket().

```
EFI_STATUS Ikev2EncodePacket ( IN IKEV2_SESSION_COMMON *   SessionCommon,
                               IN OUT IKE_PACKET *         IkePacket,
                               IN UINTN                    IkeType
                             )
```

Encode the IKE packet.

This function puts all Payloads into one payload then encrypt it if needed.

Parameters:

`[in]`	*SessionCommon*	Pointer to IKEV2_SESSION_COMMON containing some parameter used during IKE packet encoding.
`[in,out]`	*IkePacket*	Pointer to IKE_PACKET to be encoded as input, and the encoded result as output.
`[in]`	*IkeType*	The type of IKE. IKE_SA_TYPE, IKE_INFO_TYPE and IKE_CHILD_TYPE are supportted.

Return values:

EFI_SUCCESS	Encode IKE packet successfully.
Otherwise	Encode IKE packet failed.

References IKE_PAYLOAD_BY_PACKET, IkeHdrHostToNet(), IkeStateAuth, IKEV2_EXCHANGE_TYPE_INIT, IKEV2_SA_SESSION_FROM_COMMON, Ikev2EncodePayload(), Ikev2EncryptPacket(), IKEV2_SA_SESSION::InitPacket, IKEV2_SA_SESSION::InitPacketSize, IKE_PAYLOAD::PayloadBuf, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, IKEV2_SA_SESSION::RespPacket, and IKEV2_SA_SESSION::RespPacketSize.

Referenced by IkeNetbufFromPacket().

```
EFI_STATUS Ikev2EncodePayload ( IN UINT8 *                 SessionCommon,
                                IN OUT IKE_PAYLOAD *       IkePayload
                              )
```

General interface of payload encoding.

This function encodes the internal data structure into payload which is defined in RFC 4306. The IkePayload->PayloadBuf is used to store both the input payload and converted payload. Only the SA payload use the interal structure to store the attribute. Other payload use structure which is same with the RFC defined, for this kind payloads just do host order to network order change of some fields.

Parameters:

[in]	*SessionCommon*	Pointer to IKE Session Common used to encode the payload.
[in,out]	*IkePayload*	Pointer to IKE payload to be encoded as input, and store the encoded result as output.

Return values:

EFI_INVALID_PARAMETER	Meet error when encoding the SA payload.
EFI_SUCCESS	Encoded successfully.

References IKEV2_CFG_ATTRIBUTES::AttritType, IKEV2_KEY_EXCHANGE::DhGroup, TRAFFIC_SELECTOR::EndPort, IKEV2_DUMP_PAYLOAD, IKEV2_PAYLOAD_TYPE_AUTH, IKEV2_PAYLOAD_TYPE_CP, IKEV2_PAYLOAD_TYPE_DELETE, IKEV2_PAYLOAD_TYPE_ID_INIT, IKEV2_PAYLOAD_TYPE_ID_RSP, IKEV2_PAYLOAD_TYPE_KE, IKEV2_PAYLOAD_TYPE_NOTIFY, IKEV2_PAYLOAD_TYPE_SA, IKEV2_PAYLOAD_TYPE_TS_INIT, IKEV2_PAYLOAD_TYPE_TS_RSP, Ikev2EncodeSa(), IKEV2_NOTIFY::MessageType, IKEV2_DELETE::NumSpis, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, TRAFFIC_SELECTOR::SelecorLen, TRAFFIC_SELECTOR::StartPort, IKEV2_TS::TSNumbers, and IKEV2_CFG_ATTRIBUTES::ValueLength.

Referenced by Ikev2EncodePacket().

```
IKEV2_SA* Ikev2EncodeSa ( IN IKEV2_SESSION_COMMON *   SessionCommon,
                          IN IKEV2_SA_DATA *          SaData
                        )
```

Transfer the intrnal data structure IKEV2_SA_DATA to IKEV2_SA structure defined in RFC.

Parameters:

[in]	*SessionCommon*	Pointer to IKEV2_SESSION_COMMON related to the SA Session.
[in]	*SaData*	Pointer to IKEV2_SA_DATA to be transfered.

Return values:

return	the pointer of IKEV2_SA.

References IKE_SA_ATTRIBUTE::Attr, IKEV2_TRANSFORM_DATA::Attribute, IKE_SA_ATTRIBUTE::AttrType, IKE_SA_ATTR_UNION::AttrValue, IKEV2_PROPOSAL::Header, IKEV2_TRANSFORM::Header, IKEV2_SA::Header, IKE_PROPOSAL_NEXT_PAYLOAD_MORE, IKE_PROPOSAL_NEXT_PAYLOAD_NONE, IKE_TRANSFORM_NEXT_PAYLOAD_MORE, IKE_TRANSFORM_NEXT_PAYLOAD_NONE, IKEV2_ATTRIBUTE_TYPE_KEYLEN, IKEV2_TRANSFORM_TYPE_ENCR, IKEV2_TRANSFORM_TYPE_INTEG, MAX_SA_ATTRS_SIZE, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKEV2_PROPOSAL::NumTransforms, IKEV2_PROPOSAL_DATA::NumTransforms, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKEV2_PROPOSAL_DATA::ProposalIndex, IKEV2_PROPOSAL::ProposalIndex, IKEV2_PROPOSAL_DATA::ProtocolId, IKEV2_PROPOSAL::ProtocolId, SA_ATTR_FORMAT_BIT, IKEV2_PROPOSAL_DATA::Spi, IKEV2_PROPOSAL::SpiSize, IKEV2_TRANSFORM_DATA::TransformId, IKEV2_TRANSFORM::TransformId, IKEV2_TRANSFORM_DATA::TransformType, and IKEV2_TRANSFORM::TransformType.

Referenced by Ikev2EncodePayload().

```
EFI_STATUS Ikev2EncryptPacket ( IN IKEV2_SESSION_COMMON *   SessionCommon,
```

IN OUT IKE_PACKET * *IkePacket*
)

Encrypt IKE packet.

This function encrypt IKE packet before sending it. The Encrypted IKE packet is put in to IKEV2 Encrypted Payload.

Parameters:

[in]	*SessionCommon*	Pointer to IKEV2_SESSION_COMMON related to the IKE packet.
[in,out]	*IkePacket*	Pointer to IKE packet to be encrypted.

Return values:

EFI_SUCCESS	Operation is successful.
Others	Operation is failed.

References ClearAllPayloads(), HASH_DATA_FRAGMENT::Data, HASH_DATA_FRAGMENT::DataSize, IKEV2_SA_PARAMS::EncAlgId, IKE_PACKET_APPEND_PAYLOAD, IKE_PAYLOAD_BY_PACKET, IkeGenerateIv(), IkeHdrHostToNet(), IKEV2_SA_SESSION::IkeKeys, IkePayloadAlloc(), IKEV2_CHILD_SA_SESSION::IkeSaSession, IkeSessionTypeChildSa, IkeSessionTypeIkeSa, IKEV2_CHILD_SA_SESSION_FROM_COMMON, IKEV2_PAYLOAD_TYPE_ENCRYPT, IKEV2_SA_SESSION_FROM_COMMON, IKEV2_SA_PARAMS::IntegAlgId, IPSEC_DUMP_BUF, IpSecCryptoIoEncrypt(), IpSecCryptoIoHmac(), IpSecGetEncryptBlockSize(), IpSecGetIcvLength(), IKE_PAYLOAD::PayloadBuf, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, IKEV2_SESSION_COMMON::SaParams, IKEV2_SA_SESSION::SessionCommon, IKEV2_SESSION_KEYS::SkAiKey, IKEV2_SESSION_KEYS::SkAiKeySize, IKEV2_SESSION_KEYS::SkArKey, IKEV2_SESSION_KEYS::SkArKeySize, IKEV2_SESSION_KEYS::SkEiKey, IKEV2_SESSION_KEYS::SkEiKeySize, IKEV2_SESSION_KEYS::SkErKey, and IKEV2_SESSION_KEYS::SkErKeySize.

Referenced by Ikev2EncodePacket().

IKE_PAYLOAD* Ikev2GenerateCertIdPayload (IN IKEV2_SESSION_COMMON * *CommonSession*,
IN UINT8 *NextPayload*,
IN UINT8 * *InCert*,
IN UINTN *CertSize*
)

Generate a ID payload.

Parameters:

[in]	*CommonSession*	Pointer to IKEV2_SESSION_COMMON related to ID payload.
[in]	*NextPayload*	The payload type presented in the NextPayload field of ID Payload header.
[in]	*InCert*	Pointer to the Certificate which distinguished name will be added into the Id payload.
[in]	*CertSize*	Size of the Certificate.

Return values:

Pointer to ID IKE payload.

References IKEV2_ID::Header, IKEV2_ID::IdType, IkePayloadAlloc(), IKEV2_PAYLOAD_TYPE_ID_INIT, IKEV2_PAYLOAD_TYPE_ID_RSP, IpSecCryptoIoGetSubjectFromCert(), IKEV2_COMMON_PAYLOAD_HEADER::NextPayload,

IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2AuthCertGenerator().

```
IKE_PAYLOAD* Ikev2GenerateCertificatePayload ( IN IKEV2_SA_SESSION * IkeSaSession,
                                               IN UINT8 NextPayload,
                                               IN UINT8 * Certificate,
                                               IN UINTN CertificateLen,
                                               IN UINT8 EncodeType,
                                               IN BOOLEAN IsRequest )
```

Generate the Certificate payload or Certificate Request Payload.

Since the Certificate Payload structure is same with Certificate Request Payload, the only difference is that one contains the Certificate Data, other contains the acceptable certificateion CA. This function generate Certificate payload or Certificate Request Payload defined in RFC 4306, but all the fields in the payload are still in host order and need call Ikev2EncodePayload() to convert those fields from the host order to network order beforing sending it.

Parameters:

[in]	*IkeSaSession*	Pointer to IKE SA Session to be used of Delete payload generation.
[in]	*NextPayload*	The next paylaod type in NextPayload field of the Delete payload.
[in]	*Certificate*	Pointer of buffer contains the certification data.
[in]	*CertificateLen*	The length of Certificate in byte.
[in]	*EncodeType*	Specified the Certificate Encodeing which is defined in RFC 4306.
[in]	*IsRequest*	To indicate create Certificate Payload or Certificate Request Payload. If it is TURE, create Certificate Payload. Otherwise, create Certificate Request Payload.

Return values:

a Pointer to IKE Payload whose payload buffer containing the Certificate payload or Certificated Request payload.

References IKEV2_CERT::CertEncoding, HASH_DATA_FRAGMENT::Data, HASH_DATA_FRAGMENT::DataSize, IKEV2_CERT::Header, IKE_AALG_SHA1HMAC, IkePayloadAlloc(), IKEV2_PAYLOAD_TYPE_CERT, IKEV2_PAYLOAD_TYPE_CERTREQ, IpSecCryptoIoGetPublicKeyFromCert(), IpSecCryptoIoHash(), IpSecGetHmacDigestLength(), IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2AuthCertGenerator(), and Ikev2InitCertGenerator().

```
IKE_PAYLOAD* Ikev2GenerateCpPayload ( IN IKEV2_SA_SESSION * IkeSaSession,
                                      IN UINT8 NextPayload,
```

	IN UINT8	*CfgType*
)		

Generate the Configuration payload.

This function generate configuration payload defined in RFC 4306, but all the fields in this payload are still in host order and need call Ikev2EncodePayload() to convert those fields from the host order to network order beforing sending it.

Parameters:

[in]	*IkeSaSession*	Pointer to IKE SA Session to be used for Delete payload generation.
[in]	*NextPayload*	The next paylaod type in NextPayload field of the Delete payload.
[in]	*CfgType*	The attribute type in the Configuration attribute.

Return values:

Pointer to IKE CP Payload.

References IKEV2_CFG_ATTRIBUTES::AttritType, IKEV2_CFG::CfgType, IKEV2_CFG::Header, IkePayloadAlloc(), IKEV2_CFG_TYPE_REQUEST, IKEV2_PAYLOAD_TYPE_CP, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, and IKEV2_CFG_ATTRIBUTES::ValueLength.

Referenced by Ikev2AuthCertGenerator(), and Ikev2AuthPskGenerator().

IKE_PAYLOAD* Ikev2GenerateDeletePayload	(	IN IKEV2_SA_SESSION *	*IkeSaSession*,
		IN UINT8	*NextPayload*,
		IN UINT8	*SpiSize*,
		IN UINT16	*SpiNum*,
		IN UINT8 *	*SpiBuf*
	)		

Generate the Delete payload.

Since the structure of Delete payload which defined in RFC 4306 is simple, there is no internal data structure for Delete payload. This function generate Delete payload defined in RFC 4306, but all the fields in this payload are still in host order and need call Ikev2EncodePayload() to convert those fields from the host order to network order beforing sending it.

Parameters:

[in]	*IkeSaSession*	Pointer to IKE SA Session to be used of Delete payload generation.
[in]	*NextPayload*	The next paylaod type in NextPayload field of the Delete payload.
[in]	*SpiSize*	Size of the SPI in SPI size field of the Delete Payload.
[in]	*SpiNum*	Number of SPI in NumofSPIs field of the Delete Payload.
[in]	*SpiBuf*	Pointer to buffer contains the SPI value.

Return values:

a Pointer of IKE Delete Payload.

References IKEV2_DELETE::Header, IkePayloadAlloc(), IKEV2_PAYLOAD_TYPE_DELETE, IPSEC_PROTO_IPSEC_ESP, IPSEC_PROTO_ISAKMP,

IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKEV2_DELETE::NumSpis, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, IKEV2_DELETE::ProtocolId, and IKEV2_DELETE::SpiSize.

Referenced by Ikev2InfoGenerator().

```
IKE_PAYLOAD* Ikev2GenerateIdPayload ( IN IKEV2_SESSION_COMMON *   CommonSession,
                                      IN UINT8                    NextPayload
                                    )
```

Generate a ID payload.

Parameters:

[in]	*CommonSession*	Pointer to IKEV2_SESSION_COMMON related to ID payload.
[in]	*NextPayload*	The payload type presented in the NextPayload field of ID Payload header.

Return values:

Pointer to ID IKE payload.

References IKEV2_ID::Header, IKEV2_ID::IdType, IkePayloadAlloc(), IKEV2_ID_TYPE_IPV4_ADDR, IKEV2_ID_TYPE_IPV6_ADDR, IKEV2_PAYLOAD_TYPE_ID_INIT, IKEV2_PAYLOAD_TYPE_ID_RSP, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2AuthPskGenerator().

```
IKE_PAYLOAD* Ikev2GenerateKePayload ( IN OUT IKEV2_SA_SESSION *   IkeSaSession,
                                      IN UINT8                    NextPayload
                                    )
```

Generate a Key Exchange payload according to the DH group type and save the public Key into IkeSaSession IkeKey field.

Parameters:

[in,out]	*IkeSaSession*	Pointer of the IKE_SA_SESSION.
[in]	*NextPayload*	The payload type presented in the NextPayload field of Key Exchange Payload header.

Return values:

Pointer to Key IKE payload.

References IKEV2_SESSION_KEYS::DhBuffer, IKEV2_KEY_EXCHANGE::DhGroup, IKEV2_DH_BUFFER::GxBuffer, IKEV2_DH_BUFFER::GxSize, IKEV2_KEY_EXCHANGE::Header, IkePayloadAlloc(), IKEV2_PAYLOAD_TYPE_KE, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2InitPskGenerator().

```
IKE_PAYLOAD* Ikev2GenerateNoncePayload ( IN UINT8 *   NonceBuf,
                                         IN UINTN     NonceSize,
                                         IN UINT8     NextPayload
                                       )
```

Generate a Nonce payload containing the input parameter NonceBuf.

Parameters:

[in]	*NonceBuf*	The nonce buffer contains the whole Nonce payload block except the payload header.
[in]	*NonceSize*	The buffer size of the NonceBuf
[in]	*NextPayload*	The payload type presented in the NextPayload field of Nonce Payload header.

Return values:

Pointer to Nonce IKE paload.

References IKEV2_NONCE::Header, IkePayloadAlloc(), IKEV2_PAYLOAD_TYPE_NONCE, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2InitPskGenerator().

```
IKE_PAYLOAD* Ikev2GenerateNotifyPayload ( IN UINT8     ProtocolId,
                                          IN UINT8     NextPayload,
                                          IN UINT8     SpiSize,
                                          IN UINT16    MessageType,
                                          IN UINT8 *   SpiBuf,
                                          IN UINT8 *   NotifyData,
                                          IN UINTN     NotifyDataSize
                                        )
```

Generate the Notify payload.

Since the structure of Notify payload which defined in RFC 4306 is simple, so there is no internal data structure for Notify payload. This function generate Notify payload defined in RFC 4306, but all the fields in this payload are still in host order and need call Ikev2EncodePayload() to convert those fields from the host order to network order beforing sending it.

Parameters:

[in]	*ProtocolId*	The protocol type ID. For IKE_SA it MUST be one (1). For IPsec SAs it MUST be neither (2) for AH or (3) for ESP.
[in]	*NextPayload*	The next paylaod type in NextPayload field of the Notify payload.
[in]	*SpiSize*	Size of the SPI in SPI size field of the Notify Payload.
[in]	*MessageType*	The message type in NotifyMessageType field of the Notify Payload.
[in]	*SpiBuf*	Pointer to buffer contains the SPI value.
[in]	*NotifyData*	Pointer to buffer contains the notification data.
[in]	*NotifyDataSize*	The size of NotifyData in bytes.

Return values:

Pointer to IKE Notify Payload.

References IKEV2_NOTIFY::Header, IkePayloadAlloc(), IKEV2_PAYLOAD_TYPE_NOTIFY, IKEV2_NOTIFY::MessageType, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, IKEV2_NOTIFY::ProtocolId, and IKEV2_NOTIFY::SpiSize.

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), and Ikev2InitPskGenerator().

```
IKE_PAYLOAD* Ikev2GenerateSaPayload ( IN IKEV2_SA_DATA *      SessionSaData,
                                      IN UINT8                NextPayload,
                                      IN IKE_SESSION_TYPE     Type
                                    )
```

Generate Ikev2 SA payload according to SessionSaData

Parameters:

[in]	*SessionSaData*	The data used in SA payload.
[in]	*NextPayload*	The payload type presented in NextPayload field of SA Payload header.
[in]	*Type*	The SA type. It MUST be neither (1) for IKE_SA or (2) for CHILD_SA or (3) for INFO.

Return values:

a Pointer to SA IKE payload.

References IKEV2_SA::Header, IkePayloadAlloc(), IkeSessionTypeIkeSa, IKEV2_PAYLOAD_TYPE_SA, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKE_PAYLOAD::PayloadType, and IKEV2_SA_DATA::SaHeader.

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), and Ikev2InitPskGenerator().

```
IKE_PAYLOAD* Ikev2GenerateTsPayload ( IN IKEV2_CHILD_SA_SESSION *   ChildSa,
                                      IN UINT8                      NextPayload,
                                      IN BOOLEAN                    IsTunnel
                                    )
```

Generate TS payload.

This function generates TSi or TSr payload according to type of next payload. If the next payload is Responder TS, gereate TSi Payload. Otherwise, generate TSr payload.

Parameters:

[in]	*ChildSa*	Pointer to IKEV2_CHILD_SA_SESSION related to this TS payload.
[in]	*NextPayload*	The payload type presented in the NextPayload field of ID Payload header.
[in]	*IsTunnel*	It indicates that if the Ts Payload is after the CP payload. If yes, it means the Tsi and Tsr payload should be with Max port range and address range and protocol is marked as zero.

Return values:

Pointer to Ts IKE payload.

References TRAFFIC_SELECTOR::EndPort, IKEV2_TS::Header, IkePayloadAlloc(), IkePayloadFree(), IKEV2_PAYLOAD_TYPE_TS_INIT, IKEV2_PAYLOAD_TYPE_TS_RSP, IKEV2_TS_ANY_PORT, IKEV2_TS_ANY_PROTOCOL, IKEV2_TS_TYPE_IPV4_ADDR_RANGE, IKEV2_TS_TYPS_IPV6_ADDR_RANGE, TRAFFIC_SELECTOR::IpProtocolId, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, TRAFFIC_SELECTOR::SelecorLen, TRAFFIC_SELECTOR::StartPort, IKEV2_TS::TSNumbers, and TRAFFIC_SELECTOR::TSType.

Referenced by Ikev2AuthCertGenerator(), and Ikev2AuthPskGenerator().

```
VOID Ikev2OnPacketAccepted ( IN IKEV2_SESSION_COMMON *   SessionCommon,
                             IN IKE_PACKET *             IkePacket,
                             IN UINT8                    IkeType
                           )
```

Save some useful payloads after accepting the Packet.

Parameters:

[in]	*SessionCommon*	Pointer to IKEV2_SESSION_COMMON related to the operation.
[in]	*IkePacket*	Pointer to received IkePacet.
[in]	*IkeType*	The type used to indicate it is in IkeSa or ChildSa or Info exchange.

```
VOID EFIAPI Ikev2OnPacketSent ( IN NET_BUF *          Packet,
                                IN UDP_END_POINT *    EndPoint,
                                IN EFI_STATUS         IoStatus,
                                IN VOID *             Context
                              )
```

The notification function. It will be called when the related UDP_TX_TOKEN's event is signaled.

This function frees the Net Buffer pointed to the input Packet.

Parameters:

[in]	*Packet*	Pointer to Net buffer containing the sending IKE packet.
[in]	*EndPoint*	Pointer to UDP_END_POINT containing the remote and local address information.
[in]	*IoStatus*	The Status of the related UDP_TX_TOKEN.
[in]	*Context*	Pointer to data passed from the caller.

References IKEV2_SA_SESSION::BySessionTable, IKEV2_SA_SESSION::ChildSaEstablishSessionList, IKEV2_SA_SESSION::DeleteSaList, IkePacketFree(), _IPSEC_PRIVATE_DATA::Ikev1EstablishedList, IKEV2_DELET_CHILDSA_LIST, Ikev2ChildSaSessionLookupBySpi(), Ikev2ChildSaSessionRemove(), Ikev2ChildSaSilentDelete(), _IPSEC_PRIVATE_DATA::Ikev2EstablishedList, Ikev2SaSessionFree(), Ikev2SaSessionLookup(), IKEV2_SA_SESSION::InitiatorCookie, _IPSEC_PRIVATE_DATA::IpSec, IPSEC_STATUS_DISABLED, IPSECCONFIG_STATUS_NAME, IKE_PACKET::IsDeleteInfo, _IPSEC_PRIVATE_DATA::IsIPsecDisabling, IKEV2_CHILD_SA_SESSION::LocalPeerSpi, IKE_PACKET::Private, IKE_PACKET::RemotePeerIp, IKEV2_SA_SESSION::ResponderCookie, and IKE_PACKET::Spi.

Referenced by Ikev2SendIkePacket().

```
EFI_STATUS Ikev2ParserNotifyCookiePayload ( IN IKE_PAYLOAD *          IkeNCookie,
                                            IN OUT IKEV2_SA_SESSION * IkeSaSession
                                          )
```

Parser the Notify Cookie payload.

This function parses the Notify Cookie payload.If the Notify ProtocolId is not IPSEC_PROTO_ISAKMP or if the SpiSize is not zero or if the MessageType is not the COOKIE, return EFI_INVALID_PARAMETER.

Parameters:

[in]	*IkeNCookie*	Pointer to the IKE_PAYLOAD which contians the Notify Cookie payload. the Notify payload.
[in,out]	*IkeSaSession*	Pointer to the relevant IKE SA Session.

Return values:

EFI_SUCCESS	The Notify Cookie Payload is valid.
EFI_INVALID_PARAMETER	The Notify Cookie Payload is invalid.
EFI_OUT_OF_RESOURCE	The required resource can't be allocated.

References IKEV2_NOTIFY::Header, IKEV2_NOTIFICATION_COOKIE, IPSEC_PROTO_ISAKMP, IKEV2_NOTIFY::MessageType, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKEV2_NOTIFY::ProtocolId, and IKEV2_NOTIFY::SpiSize.

Referenced by Ikev2InitPskParser().

```
IKE_PAYLOAD* Ikev2PskGenerateAuthPayload ( IN IKEV2_SA_SESSION * IkeSaSession,
                                           IN IKE_PAYLOAD *      IdPayload,
                                           IN UINT8              NextPayload,
                                           IN BOOLEAN            IsVerify
                                         )
```

Generate a Authentication Payload.

This function is used for both Authentication generation and verification. When the IsVerify is TRUE, it create a Auth Data for verification. This function choose the related IKE_SA_INIT Message for Auth data creation according to the IKE Session's type and the value of IsVerify parameter.

Parameters:

[in]	*IkeSaSession*	Pointer to IKEV2_SA_SESSION related to.
[in]	*IdPayload*	Pointer to the ID payload to be used for Authentication payload generation.
[in]	*NextPayload*	The type filled into the Authentication Payload next payload field.
[in]	*IsVerify*	If it is TURE, the Authentication payload is used for verification.

Returns:

pointer to IKE Authentication payload for Pre-shared key method.

References IKEV2_AUTH::AuthMethod, CONSTANT_KEY_SIZE, HASH_DATA_FRAGMENT::Data, PRF_DATA_FRAGMENT::Data, HASH_DATA_FRAGMENT::DataSize, PRF_DATA_FRAGMENT::DataSize, IKEV2_AUTH::Header, IkePayloadAlloc(), IkePayloadFree(), IKEV2_AUTH_METHOD_SKMI, IKEV2_PAYLOAD_TYPE_AUTH, IpSecCryptoIoHmac(), IpSecGetHmacDigestLength(), mConstantKey, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload,

IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2AuthPskGenerator(), and Ikev2AuthPskParser().

```
EFI_STATUS Ikev2SendIkePacket ( IN IKE_UDP_SERVICE *   IkeUdpService,
                                IN UINT8 *             SessionCommon,
                                IN IKE_PACKET *        IkePacket,
                                IN UINTN               IkeType
                              )
```

Send out IKEV2 packet.

Parameters:

[in]	*IkeUdpService*	Pointer to IKE_UDP_SERVICE used to send the IKE packet.
[in]	*SessionCommon*	Pointer to IKEV1_SESSION_COMMON related to the IKE packet.
[in]	*IkePacket*	Pointer to IKE_PACKET to be sent out.
[in]	*IkeType*	The type of IKE to point what's kind of the IKE packet is to be sent out. IKE_SA_TYPE, IKE_INFO_TYPE and IKE_CHILD_TYPE are supportted.

Return values:

EFI_SUCCESS	The operation complete successfully.
Otherwise	The operation is failed.

References IKE_DEFAULT_PORT, IKE_DEFAULT_TIMEOUT_INTERVAL, IKE_HEADER_FLAGS_INIT, IKE_PACKET_REF, IkeNetbufFromPacket(), IkePacketFree(), Ikev2OnPacketSent(), IPSEC_DUMP_PACKET, IKEV2_SESSION_COMMON::LastSentPacket, IKEV2_SESSION_COMMON::LocalPeerIp, IKEV2_SESSION_COMMON::RemotePeerIp, IKEV2_SESSION_COMMON::TimeoutEvent, and IKEV2_SESSION_COMMON::TimeoutInterval.

Referenced by Ikev2HandleChildSa(), Ikev2HandleSa(), Ikev2InfoParser(), Ikev2NegotiateChildSa(), Ikev2NegotiateInfo(), Ikev2NegotiateSa(), and Ikev2ResendNotify().

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED CHAR8 mConstantKey[CONSTANT_KEY_SIZE]

Initial value:

```
{
  'K', 'e', 'y', ' ', 'P', 'a', 'd', ' ', 'f', 'o', 'r', ' ', 'I', 'K', 'E', 'v', '2'
}
```

Referenced by Ikev2PskGenerateAuthPayload().

NetworkPkg/IpSecDxe/Ikev2/Payload.h File Reference

Data Structures

struct IKEV2_COMMON_PAYLOAD_HEADE
struct IKEV2_SA
struct IKEV2_PROPOSAL
struct IKEV2_TRANSFORM
struct IKEV2_KEY_EXCHANGE
struct IKEV2_ID
struct IKEV2_CERT
struct IKEV2_CERT_REQ
struct IKEV2_AUTH
struct IKEV2_NONCE
struct IKEV2_NOTIFY
struct IKEV2_DELETE
struct IKEV2_TS
struct TRAFFIC_SELECTOR
struct IKEV2_VENDOR
struct IKEV2_ENCRYPTED
struct IKEV2_PAD_LEN
struct IKEV2_CFG
struct IKEV2_CFG_ATTRIBUTES

Defines

"memItemRight" valign="bottom">IKEV2_PAYLOAD_TYPE_NONE 0
#define IKEV2_PAYLOAD_TYPE_SA 33
#define IKEV2_PAYLOAD_TYPE_KE 34
#define IKEV2_PAYLOAD_TYPE_ID_INIT 35
#define IKEV2_PAYLOAD_TYPE_ID_RSP 36
#define IKEV2_PAYLOAD_TYPE_CERT 37
#define IKEV2_PAYLOAD_TYPE_CERTREQ
#define IKEV2_PAYLOAD_TYPE_AUTH 39
#define IKEV2_PAYLOAD_TYPE_NONCE 40
#define IKEV2_PAYLOAD_TYPE_NOTIFY 41
#define IKEV2_PAYLOAD_TYPE_DELETE 42
#define IKEV2_PAYLOAD_TYPE_VENDOR 4
#define IKEV2_PAYLOAD_TYPE_TS_INIT 44
#define IKEV2_PAYLOAD_TYPE_TS_RSP 45
#define IKEV2_PAYLOAD_TYPE_ENCRYPT
#define IKEV2_PAYLOAD_TYPE_CP 47
#define IKEV2_PAYLOAD_TYPE_EAP 48
#define IKE_HEADER_FLAGS_INIT 0x08
#define IKE_HEADER_FLAGS_RESPOND 0x2
#define IKE_HEADER_FLAGS_CHILD_INIT 0
#define IKEV2_EXCHANGE_TYPE_INIT 34
#define IKEV2_EXCHANGE_TYPE_AUTH 35
#define IKEV2_EXCHANGE_TYPE_CREATE_C
#define IKEV2_EXCHANGE_TYPE_INFO 37

```
#define IKEV2_TRANSFORM_TYPE_ENCR 1
#define IKEV2_TRANSFORM_TYPE_PRF 2
#define IKEV2_TRANSFORM_TYPE_INTEG 3
#define IKEV2_TRANSFORM_TYPE_DH 4
#define IKEV2_TRANSFORM_TYPE_ESN 5
#define IKEV2_TRANSFORM_ID_ENCR_DES_IV64 1
#define IKEV2_TRANSFORM_ID_ENCR_DES 2
#define IKEV2_TRANSFORM_ID_ENCR_3DES 3
#define IKEV2_TRANSFORM_ID_ENCR_RC5 4
#define IKEV2_TRANSFORM_ID_ENCR_IDEA 5
#define IKEV2_TRANSFORM_ID_ENCR_CAST 6
#define IKEV2_TRANSFORM_ID_ENCR_BLOWFISH
#define IKEV2_TRANSFORM_ID_ENCR_3IDEA 8
#define IKEV2_TRANSFORM_ID_ENCR_DES_IV32 9
#define IKEV2_TRANSFORM_ID_ENCR_NULL 11
#define IKEV2_TRANSFORM_ID_ENCR_AES_CBC 1
#define IKEV2_TRANSFORM_ID_ENCR_AES_CTR 1
#define IKEV2_TRANSFORM_ID_PRF_HMAC_MD5
#define IKEV2_TRANSFORM_ID_PRF_HMAC_SHA1
#define IKEV2_TRANSFORM_ID_PRF_HMAC_TIGER
#define IKEV2_TRANSFORM_ID_PRF_AES128_XCBC
#define IKEV2_TRANSFORM_ID_AUTH_NONE 0
#define IKEV2_TRANSFORM_ID_AUTH_HMAC_MD5
#define IKEV2_TRANSFORM_ID_AUTH_HMAC_SHA
#define IKEV2_TRANSFORM_ID_AUTH_HMAC_DES_
#define IKEV2_TRANSFORM_ID_AUTH_HMAC_KPD
#define IKEV2_TRANSFORM_ID_AUTH_HMAC_AES_
#define IKEV2_TRANSFORM_ID_DH_768MODP 1
#define IKEV2_TRANSFORM_ID_DH_1024MODP 2
#define IKEV2_TRANSFORM_ID_DH_2048MODP 14
#define IKEV2_ATTRIBUTE_TYPE_KEYLEN 14
#define IKEV2_ID_TYPE_IPV4_ADDR 1
#define IKEV2_ID_TYPE_FQDN 2
#define IKEV2_ID_TYPE_RFC822_ADDR 3
#define IKEV2_ID_TYPE_IPV6_ADDR 5
#define IKEV2_ID_TYPE_DER_ASN1_DN 9
#define IKEV2_ID_TYPE_DER_ASN1_GN 10
#define IKEV2_ID_TYPE_KEY_ID 11
#define IKEV2_CERT_ENCODEING_RESERVED 0
#define IKEV2_CERT_ENCODEING_X509_CERT_WRA
#define IKEV2_CERT_ENCODEING_PGP_CERT 2
#define IKEV2_CERT_ENCODEING_DNS_SIGN_KEY
#define IKEV2_CERT_ENCODEING_X509_CERT_SIGN
#define IKEV2_CERT_ENCODEING_KERBEROS_TOK
#define IKEV2_CERT_ENCODEING_REVOCATION_L
#define IKEV2_CERT_ENCODEING_AUTH_REVOCAT
#define IKEV2_CERT_ENCODEING_SPKI_CERT 9
#define IKEV2_CERT_ENCODEING_X509_CERT_ATT
#define IKEV2_CERT_ENCODEING_RAW_RSA_KEY
#define IKEV2_CERT_ENCODEING_HASH_AND_URL
#define IKEV2_AUTH_METHOD_RSA 1;
#define IKEV2_AUTH_METHOD_SKMI 2;
#define IKEV2_AUTH_METHOD_DSS 3;
#define IKEV2_NOTIFICATION_UNSUPPORT_CRITIC
```

#define IKEV2_NOTIFICATION_INVALID_IKE
#define IKEV2_NOTIFICATION_INVALID_MA
#define IKEV2_NOTIFICATION_INVALID_SYN
#define IKEV2_NOTIFICATION_INVALID_ME
#define IKEV2_NOTIFICATION_INVALID_SPI
#define IKEV2_NOTIFICATION_NO_PROPOSA
#define IKEV2_NOTIFICATION_INVALID_KE
#define IKEV2_NOTIFICATION_AUTHENTICA
#define IKEV2_NOTIFICATION_SINGLE_PAIR
#define IKEV2_NOTIFICATION_NO_ADDITIO
#define IKEV2_NOTIFICATION_INTERNAL_A
#define IKEV2_NOTIFICATION_FAILED_CP_F
#define IKEV2_NOTIFICATION_TS_UNCCEPT
#define IKEV2_NOTIFICATION_INVALID_SEL
#define IKEV2_NOTIFICATION_COOKIE 163
#define IKEV2_NOTIFICATION_USE_TRANSP
#define IKEV2_NOTIFICATION_REKEY_SA 1
#define IKEV2_TS_TYPE_IPV4_ADDR_RANGE
#define IKEV2_TS_TYPS_IPV6_ADDR_RANGE
#define IKEV2_CFG_TYPE_REQUEST 1
#define IKEV2_CFG_TYPE_REPLY 2
#define IKEV2_CFG_TYPE_SET 3
#define IKEV2_CFG_TYPE_ACK 4
#define IKEV2_CFG_ATTR_INTERNAL_IP4_A
#define IKEV2_CFG_ATTR_INTERNAL_IP4_N
#define IKEV2_CFG_ATTR_INTERNAL_IP4_D
#define IKEV2_CFG_ATTR_INTERNAL_IP4_N
#define IKEV2_CFG_ATTR_INTERNA_ADDRE
#define IKEV2_CFG_ATTR_INTERNAL_IP4_D
#define IKEV2_CFG_ATTR_APPLICATION_VE
#define IKEV2_CFG_ATTR_INTERNAL_IP6_A
#define IKEV2_CFG_ATTR_INTERNAL_IP6_D
#define IKEV2_CFG_ATTR_INTERNAL_IP6_N
#define IKEV2_CFG_ATTR_INTERNAL_IP6_D
#define IKEV2_CFG_ATTR_INTERNAL_IP4_SU
#define IKEV2_CFG_ATTR_SUPPORTED_ATT
#define IKEV2_CFG_ATTR_IP6_SUBNET 15

Detailed Description

The Definitions related to IKEv2 payload.

Copyright (c) 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Ikev2CreateChildGenerator().

#define IKE_HEADER_FLAGS_INIT 0x08

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), Ikev2ChildSaParseSaPayload(), Ikev2InfoGenerator(), Ikev2InfoParser(), Ikev2InitPskGenerator(), Ikev2InitPskParser(), Ikev2SaParseSaPayload(), Ikev2SendIkePacket(), and Ikev2ValidateHeader().

#define IKE_HEADER_FLAGS_RESPOND 0x20

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), Ikev2ChildSaParseSaPayload(), Ikev2CreateChildGenerator(), Ikev2HandleChildSa(), Ikev2InfoGenerator(), Ikev2InitPskGenerator(), Ikev2InitPskParser(), Ikev2SaParseSaPayload(), and Ikev2ValidateHeader().

#define IKEV2_ATTRIBUTE_TYPE_KEYLEN 14

Referenced by Ikev2DecodeSa(), Ikev2EncodeSa(), Ikev2InitializeSaData(), and Ikev2ParseProposalData().

#define IKEV2_AUTH_METHOD_DSS 3;

#define IKEV2_AUTH_METHOD_RSA 1;

Referenced by Ikev2CertGenerateAuthPayload().

#define IKEV2_AUTH_METHOD_SKMI 2;

Referenced by Ikev2PskGenerateAuthPayload().

#define IKEV2_CERT_ENCODEING_AUTH_REVOCATION_LIST 8

#define IKEV2_CERT_ENCODEING_DNS_SIGN_KEY 3

#define IKEV2_CERT_ENCODEING_HASH_AND_URL_OF_X509_CERT 12

Referenced by Ikev2AuthCertGenerator(), and Ikev2InitCertGenerator().

#define IKEV2_CERT_ENCODEING_KERBEROS_TOKEN 6

#define IKEV2_CERT_ENCODEING_PGP_CERT 2

#define IKEV2_CERT_ENCODEING_RAW_RSA_KEY 11

#define IKEV2_CERT_ENCODEING_RESERVED 0

#define IKEV2_CERT_ENCODEING_REVOCATION_LIST_CERT 7

#define IKEV2_CERT_ENCODEING_SPKI_CERT 9

#define IKEV2_CERT_ENCODEING_X509_CERT_ATTRIBUTE 10

#define IKEV2_CERT_ENCODEING_X509_CERT_SIGN 4

Referenced by Ikev2AuthCertGenerator().

#define IKEV2_CERT_ENCODEING_X509_CERT_WRAP 1

#define IKEV2_CFG_ATTR_APPLICATION_VERSION 7

#define IKEV2_CFG_ATTR_INTERNA_ADDRESS_BXPIRY 5

#define IKEV2_CFG_ATTR_INTERNAL_IP4_ADDRESS 1

Referenced by Ikev2AuthCertGenerator(), and Ikev2AuthPskGenerator().

#define IKEV2_CFG_ATTR_INTERNAL_IP4_DHCP 6

#define IKEV2_CFG_ATTR_INTERNAL_IP4_DNS 3

#define IKEV2_CFG_ATTR_INTERNAL_IP4_NBNS 4

#define IKEV2_CFG_ATTR_INTERNAL_IP4_NBTMASK 2

#define IKEV2_CFG_ATTR_INTERNAL_IP4_SUBNET 13

#define IKEV2_CFG_ATTR_INTERNAL_IP6_ADDRESS 8

Referenced by Ikev2AuthCertGenerator(), and Ikev2AuthPskGenerator().

#define IKEV2_CFG_ATTR_INTERNAL_IP6_DHCP 12

#define IKEV2_CFG_ATTR_INTERNAL_IP6_DNS 10

#define IKEV2_CFG_ATTR_INTERNAL_IP6_NBNS 11

#define IKEV2_CFG_ATTR_IP6_SUBNET 15

#define IKEV2_CFG_ATTR_SUPPORTED_ATTRIBUTES 14

#define IKEV2_CFG_TYPE_ACK 4

#define IKEV2_CFG_TYPE_REPLY 2

#define IKEV2_CFG_TYPE_REQUEST 1

Referenced by Ikev2GenerateCpPayload().

#define IKEV2_CFG_TYPE_SET 3

#define IKEV2_EXCHANGE_TYPE_AUTH 35

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), and Ikev2AuthPskParser().

#define IKEV2_EXCHANGE_TYPE_CREATE_CHILD 36

Referenced by Ikev2ValidateHeader().

#define IKEV2_EXCHANGE_TYPE_INFO 37

Referenced by Ikev2DecodePacket(), Ikev2InfoGenerator(), and Ikev2ValidateHeader().

#define IKEV2_EXCHANGE_TYPE_INIT 34

Referenced by Ikev2DecodePacket(), Ikev2EncodePacket(), and Ikev2InitPskGenerator().

#define IKEV2_ID_TYPE_DER_ASN1_DN 9

#define IKEV2_ID_TYPE_DER_ASN1_GN 10

#define IKEV2_ID_TYPE_FQDN 2

#define IKEV2_ID_TYPE_IPV4_ADDR 1

Referenced by Ikev2GenerateIdPayload().

#define IKEV2_ID_TYPE_IPV6_ADDR 5

Referenced by Ikev2GenerateIdPayload().

#define IKEV2_ID_TYPE_KEY_ID 11

#define IKEV2_ID_TYPE_RFC822_ADDR 3

#define IKEV2_NOTIFICATION_AUTHENTICATION_FAILED 24

#define IKEV2_NOTIFICATION_COOKIE 16390

Referenced by Ikev2InitPskGenerator(), and Ikev2ParserNotifyCookiePayload().

#define IKEV2_NOTIFICATION_FAILED_CP_REQUIRED 37

#define IKEV2_NOTIFICATION_INTERNAL_ADDRESS_FAILURE 36

#define IKEV2_NOTIFICATION_INVALID_IKE_SPI 4

#define IKEV2_NOTIFICATION_INVALID_KEY_PAYLOAD 17

#define IKEV2_NOTIFICATION_INVALID_MAJOR_VERSION 5

#define IKEV2_NOTIFICATION_INVALID_MESSAGE_ID 9

#define IKEV2_NOTIFICATION_INVALID_SELECTORS 39

#define IKEV2_NOTIFICATION_INVALID_SPI 11

#define IKEV2_NOTIFICATION_INVALID_SYNTAX 7

#define IKEV2_NOTIFICATION_NO_ADDITIONAL_SAS 35

Referenced by Ikev2CreateChildGenerator().

#define IKEV2_NOTIFICATION_NO_PROPOSAL_CHOSEN 14

#define IKEV2_NOTIFICATION_REKEY_SA 16393

Referenced by Ikev2ChildExchangeRequestType().

#define IKEV2_NOTIFICATION_SINGLE_PAIR_REQUIRED 34

#define IKEV2_NOTIFICATION_TS_UNCCEPTABLE 38

#define IKEV2_NOTIFICATION_UNSUPPORT_CRITICAL_PAYLOAD 1

#define IKEV2_NOTIFICATION_USE_TRANSPORT_MODE 16391

Referenced by Ikev2AuthCertGenerator(), and Ikev2AuthPskGenerator().

#define IKEV2_PAYLOAD_TYPE_AUTH 39

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), Ikev2CertGenerateAuthPayload(), Ikev2DecodePayload(), Ikev2EncodePayload(), and Ikev2PskGenerateAuthPayload().

#define IKEV2_PAYLOAD_TYPE_CERT 37

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), and Ikev2GenerateCertificatePayload().

#define IKEV2_PAYLOAD_TYPE_CERTREQ 38

Referenced by Ikev2AuthCertGenerator(), Ikev2GenerateCertificatePayload(), and Ikev2InitCertGenerator().

#define IKEV2_PAYLOAD_TYPE_CP 47

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2DecodePayload(), Ikev2EncodePayload(), and Ikev2GenerateCpPayload().

#define IKEV2_PAYLOAD_TYPE_DELETE 42

Referenced by Ikev2DecodePayload(), Ikev2EncodePayload(), Ikev2GenerateDeletePayload(), Ikev2InfoGenerator(), and Ikev2InfoParser().

#define IKEV2_PAYLOAD_TYPE_EAP 48

#define IKEV2_PAYLOAD_TYPE_ENCRYPT 46

Referenced by Ikev2DecryptPacket(), and Ikev2EncryptPacket().

#define IKEV2_PAYLOAD_TYPE_ID_INIT 35

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), Ikev2DecodePayload(), Ikev2EncodePayload(), Ikev2GenerateCertIdPayload(), and Ikev2GenerateIdPayload().

#define IKEV2_PAYLOAD_TYPE_ID_RSP 36

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), Ikev2DecodePayload(), Ikev2EncodePayload(), Ikev2GenerateCertIdPayload(), and Ikev2GenerateIdPayload().

#define IKEV2_PAYLOAD_TYPE_KE 34

Referenced by Ikev2DecodePayload(), Ikev2EncodePayload(), Ikev2GenerateKePayload(), Ikev2InitPskGenerator(), and Ikev2InitPskParser().

#define IKEV2_PAYLOAD_TYPE_NONCE 40

Referenced by Ikev2GenerateNoncePayload(), Ikev2InitPskGenerator(), and Ikev2InitPskParser().

#define IKEV2_PAYLOAD_TYPE_NONE 0

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), Ikev2DecodePacket(), Ikev2InfoGenerator(), Ikev2InfoParser(), Ikev2InitCertGenerator(), and Ikev2InitPskGenerator().

#define IKEV2_PAYLOAD_TYPE_NOTIFY 41

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2ChildExchangeRequestType(), Ikev2CreateChildGenerator(), Ikev2DecodePayload(), Ikev2EncodePayload(), Ikev2GenerateNotifyPayload(), Ikev2InitPskGenerator(), and Ikev2InitPskParser().

#define IKEV2_PAYLOAD_TYPE_SA 33

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), Ikev2DecodePayload(), Ikev2EncodePayload(), Ikev2GenerateSaPayload(), Ikev2InitPskGenerator(), and Ikev2InitPskParser().

#define IKEV2_PAYLOAD_TYPE_TS_INIT 44

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), Ikev2ChildExchangeRequestType(), Ikev2DecodePayload(), Ikev2EncodePayload(), and Ikev2GenerateTsPayload().

#define IKEV2_PAYLOAD_TYPE_TS_RSP 45

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthCertParser(), Ikev2AuthPskGenerator(), Ikev2AuthPskParser(), Ikev2DecodePayload(), Ikev2EncodePayload(), and Ikev2GenerateTsPayload().

#define IKEV2_PAYLOAD_TYPE_VENDOR 43

#define IKEV2_TRANSFORM_ID_AUTH_HMAC_AES_XCBC_96 5

#define IKEV2_TRANSFORM_ID_AUTH_HMAC_DES_MAC 3

#define IKEV2_TRANSFORM_ID_AUTH_HMAC_KPDK_MD5 4

#define IKEV2_TRANSFORM_ID_AUTH_HMAC_MD5_96 1

#define IKEV2_TRANSFORM_ID_AUTH_HMAC_SHA1_96 2

Referenced by Ikev2InitializeSaData().

#define IKEV2_TRANSFORM_ID_AUTH_NONE 0

#define IKEV2_TRANSFORM_ID_DH_1024MODP 2

Referenced by Ikev2InitializeSaData(), and Ikev2NegotiateSa().

#define IKEV2_TRANSFORM_ID_DH_2048MODP 14

#define IKEV2_TRANSFORM_ID_DH_768MODP 1

#define IKEV2_TRANSFORM_ID_ENCR_3DES 3

Referenced by Ikev2InitializeSaData().

#define IKEV2_TRANSFORM_ID_ENCR_3IDEA 8

#define IKEV2_TRANSFORM_ID_ENCR_AES_CBC 12

Referenced by Ikev2InitializeSaData().

#define IKEV2_TRANSFORM_ID_ENCR_AES_CTR 13

#define IKEV2_TRANSFORM_ID_ENCR_BLOWFISH 7

#define IKEV2_TRANSFORM_ID_ENCR_CAST 6

#define IKEV2_TRANSFORM_ID_ENCR_DES 2

#define IKEV2_TRANSFORM_ID_ENCR_DES_IV32 9

#define IKEV2_TRANSFORM_ID_ENCR_DES_IV64 1

#define IKEV2_TRANSFORM_ID_ENCR_IDEA 5

#define IKEV2_TRANSFORM_ID_ENCR_NULL 11

#define IKEV2_TRANSFORM_ID_ENCR_RC5 4

#define IKEV2_TRANSFORM_ID_PRF_AES128_XCBC 4

#define IKEV2_TRANSFORM_ID_PRF_HMAC_MD5 1

#define IKEV2_TRANSFORM_ID_PRF_HMAC_SHA1 2

Referenced by Ikev2InitializeSaData().

#define IKEV2_TRANSFORM_ID_PRF_HMAC_TIGER 3

#define IKEV2_TRANSFORM_TYPE_DH 4

Referenced by Ikev2InitializeSaData(), and Ikev2ParseProposalData().

#define IKEV2_TRANSFORM_TYPE_ENCR 1

Referenced by Ikev2EncodeSa(), Ikev2InitializeSaData(), and Ikev2ParseProposalData().

#define IKEV2_TRANSFORM_TYPE_ESN 5

Referenced by Ikev2InitializeSaData(), and Ikev2ParseProposalData().

#define IKEV2_TRANSFORM_TYPE_INTEG 3

Referenced by Ikev2EncodeSa(), Ikev2InitializeSaData(), and Ikev2ParseProposalData().

#define IKEV2_TRANSFORM_TYPE_PRF 2

Referenced by Ikev2InitializeSaData(), and Ikev2ParseProposalData().

#define IKEV2_TS_TYPE_IPV4_ADDR_RANGE 7

Referenced by Ikev2GenerateTsPayload().

#define IKEV2_TS_TYPS_IPV6_ADDR_RANGE 8

Referenced by Ikev2GenerateTsPayload().

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/Ikev2/Sa.c File Reference

Functions

EFI_STATUS Ikev2GenerateSaDhPublicKey (IN IKEV2_SA_SESSION *IkeSaSession)
EFI_STATUS Ikev2GenerateSaKeys (IN IKEV2_SA_SESSION *IkeSaSession, IN IKE_PAYLOAD *KePayload)
EFI_STATUS Ikev2GenerateChildSaKeys (IN IKEV2_CHILD_SA_SESSION *ChildSaSession, IN IKE_PAYLOAD *KePayload)
IKE_PACKET * Ikev2InitPskGenerator (IN UINT8 *SaSession, IN VOID *Context)
EFI_STATUS Ikev2InitPskParser (IN UINT8 *SaSession, IN IKE_PACKET *IkePacket)
IKE_PACKET * Ikev2AuthPskGenerator (IN UINT8 *SaSession, IN VOID *Context)
EFI_STATUS Ikev2AuthPskParser (IN UINT8 *SaSession, IN IKE_PACKET *IkePacket)
IKE_PACKET * Ikev2InitCertGenerator (IN UINT8 *SaSession, IN VOID *Context)
EFI_STATUS Ikev2InitCertParser (IN UINT8 *SaSession, IN IKE_PACKET *IkePacket)
IKE_PACKET * Ikev2AuthCertGenerator (IN UINT8 *SaSession, IN VOID *Context)
EFI_STATUS Ikev2AuthCertParser (IN UINT8 *SaSession, IN IKE_PACKET *IkePacket)
EFI_STATUS Ikev2GenerateSaDhComputeKey (IN IKEV2_DH_BUFFER *DhBuffer, IN IKE_PAYLOAD *KePayload)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED
IKEV2_PACKET_HANDLER mIkev2Initial [][2]

Detailed Description

The operations for IKEv2 SA.

(C) Copyright 2015 Hewlett-Packard Development Company, L.P.
Copyright (c) 2010 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

IKE_PACKET* Ikev2AuthCertGenerator (IN UINT8 * *SaSession*,
IN VOID * *Context*
)

Generates the IKEv2 packet for IKE_AUTH exchange.

Parameters:

[in] *SaSession* Pointer to IKEV2_SA_SESSION.
[in] *Context* Context data passed by caller.

Return values:

Pointer to IKEv2 Packet to be sent out.

References IKEV2_SA_SESSION::ChildSaSessionList, _IPSEC_PAD_ENTRY::Data, _IPSEC_SPD_ENTRY::Data, IKE_HEADER::ExchangeType, IKE_HEADER::Flags, IKE_PACKET::Header, IKE_HEADER_FLAGS_INIT, IKE_HEADER_FLAGS_RESPOND, IKE_PACKET_APPEND_PAYLOAD, IkePacketAlloc(), IKEV2_CHILD_SA_SESSION::IkeSaSession, IkeSessionTypeChildSa, IKEV2_CERT_ENCODEING_HASH_AND_URL_OF_X509_CERT, IKEV2_CERT_ENCODEING_X509_CERT_SIGN, IKEV2_CFG_ATTR_INTERNAL_IP4_ADDRESS, IKEV2_CFG_ATTR_INTERNAL_IP6_ADDRESS, IKEV2_CHILD_SA_SESSION_BY_IKE_SA, IKEV2_EXCHANGE_TYPE_AUTH, IKEV2_NOTIFICATION_USE_TRANSPORT_MODE, IKEV2_PAYLOAD_TYPE_AUTH, IKEV2_PAYLOAD_TYPE_CERT, IKEV2_PAYLOAD_TYPE_CERTREQ, IKEV2_PAYLOAD_TYPE_CP, IKEV2_PAYLOAD_TYPE_ID_INIT, IKEV2_PAYLOAD_TYPE_ID_RSP, IKEV2_PAYLOAD_TYPE_NONE, IKEV2_PAYLOAD_TYPE_NOTIFY, IKEV2_PAYLOAD_TYPE_SA, IKEV2_PAYLOAD_TYPE_TS_INIT, IKEV2_PAYLOAD_TYPE_TS_RSP, Ikev2CertGenerateAuthPayload(), Ikev2GenerateCertIdPayload(), Ikev2GenerateCertificatePayload(), Ikev2GenerateCpPayload(), Ikev2GenerateNotifyPayload(), Ikev2GenerateSaPayload(), Ikev2GenerateTsPayload(), IKEV2_SA_SESSION::InitiatorCookie, IKE_HEADER::InitiatorCookie, IKE_UDP_SERVICE::IpVersion, IKEV2_SESSION_COMMON::IsInitiator, IKE_HEADER::MessageId, IKE_HEADER::NextPayload, IKEV2_SA_SESSION::Pad, _IPSEC_SPD_DATA::ProcessingPolicy, IKEV2_SA_SESSION::ResponderCookie, IKE_HEADER::ResponderCookie, IKEV2_CHILD_SA_SESSION::SaData, IKEV2_SA_SESSION::SessionCommon, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_SA_SESSION::Spd, IKEV2_SESSION_COMMON::UdpService, and IKE_HEADER::Version.

EFI_STATUS Ikev2AuthCertParser (IN UINT8 * *SaSession*,
IN IKE_PACKET * *IkePacket*
)

Parses IKE_AUTH packet.

Parameters:

[in] *SaSession* Pointer to the IKE_SA_SESSION related to this packet.
[in] *IkePacket* Pointer to the IKE_AUTH packet to be parsered.

Return values:

EFI_INVALID_PARAMETER The IKEv2 packet is malformed or the SA proposal is unacceptable.
EFI_SUCCESS The IKE packet is acceptable and the relative data is saved for furthure communication.

EFI_UNSUPPORTED The certificate authentication is not supported.

References IKEV2_SA_SESSION::ChildSaSessionList, _IPSEC_SPD_ENTRY::Data, IKE_HEADER_FLAGS_INIT, IKE_HEADER_FLAGS_RESPOND, IKE_PAYLOAD_BY_PACKET, IkePayloadFree(), IKEV2_CHILD_SA_SESSION::IkeSaSession, IkeStateIkeSaEstablished, IKEV2_CHILD_SA_SESSION_BY_IKE_SA, IKEV2_DUMP_STATE, IKEV2_EXCHANGE_TYPE_AUTH, IKEV2_PAYLOAD_TYPE_AUTH, IKEV2_PAYLOAD_TYPE_CERT, IKEV2_PAYLOAD_TYPE_ID_INIT, IKEV2_PAYLOAD_TYPE_ID_RSP, IKEV2_PAYLOAD_TYPE_SA, IKEV2_PAYLOAD_TYPE_TS_INIT, IKEV2_PAYLOAD_TYPE_TS_RSP, Ikev2CertGenerateAuthPayload(), Ikev2ChildSaAssociateSpdEntry(), Ikev2ChildSaParseSaPayload(), Ikev2ChildSaSessionSpdSelectorCreate(), Ikev2GenerateChildSaKeys(), IpSecCryptoIoVerifySignDataByCertificate(), IKE_UDP_SERVICE::IpVersion, IKEV2_SESSION_COMMON::IsInitiator, IKEV2_CHILD_SA_SESSION::LocalPort, IKE_PAYLOAD::PayloadBuf, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, _IPSEC_SPD_DATA::ProcessingPolicy, IKEV2_CHILD_SA_SESSION::ProtoId, IKEV2_CHILD_SA_SESSION::RemotePort, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_SA_SESSION::SessionCommon, IKEV2_CHILD_SA_SESSION::Spd, IKEV2_SA_SESSION::Spd, IKEV2_CHILD_SA_SESSION::SpdSelector, IKEV2_SESSION_COMMON::State, and IKEV2_SESSION_COMMON::UdpService.

```
IKE_PACKET* Ikev2AuthPskGenerator ( IN UINT8 *  SaSession,
                                    IN VOID *   Context
                                  )
```

Generates the IKEv2 packet for IKE_AUTH exchange.

Parameters:

`[in]` *SaSession* Pointer to IKEV2_SA_SESSION.

`[in]` *Context* Context data passed by caller.

Return values:

Pointer to IKE Packet to be sent out.

References IKEV2_SA_SESSION::ChildSaSessionList, _IPSEC_SPD_ENTRY::Data, IKE_HEADER::ExchangeType, IKE_HEADER::Flags, IKE_PACKET::Header, IKE_HEADER_FLAGS_INIT, IKE_HEADER_FLAGS_RESPOND, IKE_PACKET_APPEND_PAYLOAD, IkePacketAlloc(), IKEV2_CHILD_SA_SESSION::IkeSaSession, IkeSessionTypeChildSa, IKEV2_CFG_ATTR_INTERNAL_IP4_ADDRESS, IKEV2_CFG_ATTR_INTERNAL_IP6_ADDRESS, IKEV2_CHILD_SA_SESSION_BY_IKE_SA, IKEV2_EXCHANGE_TYPE_AUTH, IKEV2_NOTIFICATION_USE_TRANSPORT_MODE, IKEV2_PAYLOAD_TYPE_AUTH, IKEV2_PAYLOAD_TYPE_CP, IKEV2_PAYLOAD_TYPE_ID_INIT, IKEV2_PAYLOAD_TYPE_ID_RSP, IKEV2_PAYLOAD_TYPE_NONE, IKEV2_PAYLOAD_TYPE_NOTIFY, IKEV2_PAYLOAD_TYPE_SA, IKEV2_PAYLOAD_TYPE_TS_INIT, IKEV2_PAYLOAD_TYPE_TS_RSP, Ikev2GenerateCpPayload(), Ikev2GenerateIdPayload(), Ikev2GenerateNotifyPayload(), Ikev2GenerateSaPayload(), Ikev2GenerateTsPayload(), Ikev2PskGenerateAuthPayload(), IKEV2_SA_SESSION::InitiatorCookie, IKE_HEADER::InitiatorCookie, IKE_UDP_SERVICE::IpVersion, IKEV2_SESSION_COMMON::IsInitiator, IKE_HEADER::MessageId, IKE_HEADER::NextPayload, _IPSEC_SPD_DATA::ProcessingPolicy, IKEV2_SA_SESSION::ResponderCookie, IKE_HEADER::ResponderCookie, IKEV2_CHILD_SA_SESSION::SaData, IKEV2_SA_SESSION::SessionCommon, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_SA_SESSION::Spd, IKEV2_SESSION_COMMON::UdpService, and IKE_HEADER::Version.

```
EFI_STATUS Ikev2AuthPskParser ( IN UINT8 *          SaSession,
                                IN IKE_PACKET *     IkePacket
                              )
```

Parses IKE_AUTH packet.

Parameters:

[in] *SaSession* Pointer to the IKE_SA_SESSION related to this packet.
[in] *IkePacket* Pointer to the IKE_AUTH packet to be parsered.

Return values:

EFI_INVALID_PARAMETER The IKE packet is malformed or the SA proposal is unacceptable.
EFI_SUCCESS The IKE packet is acceptable and the relative data is saved for furthure communication.

References IKEV2_SA_SESSION::ChildSaSessionList, _IPSEC_SPD_ENTRY::Data, IKE_HEADER_FLAGS_INIT, IKE_HEADER_FLAGS_RESPOND, IKE_PAYLOAD_BY_PACKET, IKEV2_CHILD_SA_SESSION::IkeSaSession, IkeStateIkeSaEstablished, IKEV2_CHILD_SA_SESSION_BY_IKE_SA, IKEV2_DUMP_STATE, IKEV2_EXCHANGE_TYPE_AUTH, IKEV2_PAYLOAD_TYPE_AUTH, IKEV2_PAYLOAD_TYPE_ID_INIT, IKEV2_PAYLOAD_TYPE_ID_RSP, IKEV2_PAYLOAD_TYPE_SA, IKEV2_PAYLOAD_TYPE_TS_INIT, IKEV2_PAYLOAD_TYPE_TS_RSP, Ikev2ChildSaAssociateSpdEntry(), Ikev2ChildSaParseSaPayload(), Ikev2ChildSaSessionSpdSelectorCreate(), Ikev2GenerateChildSaKeys(), Ikev2PskGenerateAuthPayload(), IKE_UDP_SERVICE::IpVersion, IKEV2_SESSION_COMMON::IsInitiator, IKEV2_CHILD_SA_SESSION::LocalPort, IKE_PAYLOAD::PayloadBuf, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, _IPSEC_SPD_DATA::ProcessingPolicy, IKEV2_CHILD_SA_SESSION::ProtoId, IKEV2_CHILD_SA_SESSION::RemotePort, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_SA_SESSION::SessionCommon, IKEV2_CHILD_SA_SESSION::Spd, IKEV2_SA_SESSION::Spd, IKEV2_CHILD_SA_SESSION::SpdSelector, IKEV2_SESSION_COMMON::State, and IKEV2_SESSION_COMMON::UdpService.

```
EFI_STATUS Ikev2GenerateChildSaKeys ( IN IKEV2_CHILD_SA_SESSION *   ChildSaSession,
                                      IN IKE_PAYLOAD *              KePayload
                                    )
```

Generates the Keys for the furthure IPsec Protocol.

Parameters:

[in] *ChildSaSession* Pointer to IKE Child SA Session.
[in] *KePayload* Pointer to Key payload used to generate the Key.

Return values:

EFI_UNSUPPORTED If one or more Algorithm Id is unsupported.
EFI_SUCCESS The operation succeeded.

Generates the Keys for the furthure IPsec Protocol.

Parameters:

[in] *ChildSaSession* Pointer to IKE Child SA Session.
[in] *KePayload* Pointer to Key payload used to generate the Key.

Return values:

EFI_UNSUPPORTED If one or more Algorithm Id is not supported.
EFI_SUCCESS The operation succeeded.

References PRF_DATA_FRAGMENT::Data, PRF_DATA_FRAGMENT::DataSize, IKEV2_SA_PARAMS::EncAlgId, Ikev2GenerateSaDhComputeKey(), Ikev2SaGenerateKey(), IKEV2_SA_PARAMS::IntegAlgId, IPSEC_DUMP_BUF, IpSecGetEncryptKeyLength(), and IpSecGetHmacDigestLength().

Referenced by Ikev2AuthCertParser(), and Ikev2AuthPskParser().

```
EFI_STATUS Ikev2GenerateSaDhComputeKey ( IN IKEV2_DH_BUFFER *   DhBuffer,
                                         IN IKE_PAYLOAD *       KePayload
                                       )
```

Computes the DH Shared/Exchange Key.

Given peer's public key, this function computes the exchanged common key and stores it in the IKEv2 SA Session's GxyBuffer.

Parameters:

[in] *DhBuffer* Pointer to buffer of peer's puliic key.
[in] *KePayload* Pointer to received key payload.

Return values:

EFI_SUCCESS The operation succeeded.
Otherwise The operation failed.

References IPSEC_DUMP_BUF, and IpSecCryptoIoDhComputeKey().

Referenced by Ikev2GenerateChildSaKeys(), and Ikev2GenerateSaKeys().

```
EFI_STATUS Ikev2GenerateSaDhPublicKey ( IN IKEV2_SA_SESSION * IkeSaSession )
```

Generates the DH Key.

This generates the DH local public key and store it in the IKEv2 SA Session's GxBuffer.

Parameters:

[in] *IkeSaSession* Pointer to related IKE SA Session.

Return values:

EFI_SUCCESS The operation succeeded.
Others The operation failed.

Generates the DH Public Key.

This generates the DH local public key and store it in the IKE SA Session's GxBuffer.

Parameters:

[in] *IkeSaSession* Pointer to related IKE SA Session.

Return values:

EFI_SUCCESS The operation succeeded.
Others The operation failed.

References IKEV2_SESSION_KEYS::DhBuffer, IKEV2_DH_BUFFER::DhContext, MODP_GROUP::GroupGenerator, IKEV2_DH_BUFFER::GxBuffer, IKEV2_DH_BUFFER::GxSize, IPSEC_DUMP_BUF, IpSecCryptoIoDhGetPublicKey(), MODP_GROUP::Modulus, OakleyModpGroup, and MODP_GROUP::Size.

Referenced by Ikev2InitPskGenerator(), and Ikev2InitPskParser().

```
EFI_STATUS Ikev2GenerateSaKeys ( IN IKEV2_SA_SESSION *   IkeSaSession,
                                 IN IKE_PAYLOAD *        KePayload
                               )
```

Generates the IKEv2 SA key for the furthure IKEv2 exchange.

Parameters:

[in] *IkeSaSession* Pointer to IKEv2 SA Session.
[in] *KePayload* Pointer to Key payload used to generate the Key.

Return values:

EFI_UNSUPPORTED If the Algorithm Id is not supported.
EFI_SUCCESS The operation succeeded.

Generates the IKE SKEYSEED and seven other secrets. SK_d, SK_ai, SK_ar, SK_ei, SK_er, SK_pi, SK_pr are keys for the furthure IKE exchange.

Parameters:

[in] *IkeSaSession* Pointer to IKE SA Session.
[in] *KePayload* Pointer to Key payload used to generate the Key.

Return values:

EFI_UNSUPPORTED If one or more Algorithm Id is not supported.
EFI_OUT_OF_RESOURCES If there is no enough resource to be allocated to meet the requirement.
EFI_SUCCESS The operation succeeded.

References HASH_DATA_FRAGMENT::Data, PRF_DATA_FRAGMENT::Data, HASH_DATA_FRAGMENT::DataSize, PRF_DATA_FRAGMENT::DataSize, IKEV2_SA_PARAMS::EncAlgId, Ikev2GenerateSaDhComputeKey(), Ikev2SaGenerateKey(), IKEV2_SA_PARAMS::IntegAlgId, IPSEC_DUMP_BUF, IpSecCryptoIoHmac(), IpSecGetEncryptKeyLength(), IpSecGetHmacDigestLength(), and IKEV2_SA_PARAMS::Prf.

Referenced by Ikev2InitPskParser().

```
IKE_PACKET* Ikev2InitCertGenerator ( IN UINT8 *   SaSession,
                                     IN VOID *    Context
                                   )
```

Gernerates IKEv2 packet for IKE_SA_INIT exchange.

Parameters:

[in] *SaSession* Pointer to IKEV2_SA_SESSION related to the exchange.

[in] *Context* Context Data passed by caller.

Return values:

EFI_SUCCESS The IKE packet generation succeeded.
Others The IKE packet generation failed.

References IKE_PACKET_APPEND_PAYLOAD, IKE_PACKET_END_PAYLOAD, IKE_PAYLOAD_BY_PACKET, IKEV2_CERT_ENCODEING_HASH_AND_URL_OF_X509_CERT, IKEV2_PAYLOAD_TYPE_CERTREQ, IKEV2_PAYLOAD_TYPE_NONE, Ikev2GenerateCertificatePayload(), Ikev2InitPskGenerator(), and IKE_PAYLOAD::PayloadBuf.

```
EFI_STATUS Ikev2InitCertParser ( IN UINT8 *          SaSession,
                                 IN IKE_PACKET *     IkePacket
                               )
```

Parses the IKEv2 packet for IKE_SA_INIT exchange.

Parameters:

[in] *SaSession* Pointer to IKEV2_SA_SESSION related to the exchange.
[in] *IkePacket* The received IKEv2 packet to be parsed.

Return values:

EFI_SUCCESS The IKEv2 packet is acceptable and the relative data is saved for furthure communication.
EFI_INVALID_PARAMETER The IKE packet is malformed or the SA proposal is unacceptable.
EFI_UNSUPPORTED The certificate authentication is not supported.

References Ikev2InitPskParser().

```
IKE_PACKET* Ikev2InitPskGenerator ( IN UINT8 *   SaSession,
                                    IN VOID *    Context
                                  )
```

Gernerates IKEv2 packet for IKE_SA_INIT exchange.

Parameters:

[in] *SaSession* Pointer to IKEV2_SA_SESSION related to the exchange.
[in] *Context* Context Data passed by caller.

Return values:

EFI_SUCCESS The IKEv2 packet generation succeeded.
Others The IKEv2 packet generation failed.

References IKE_HEADER::ExchangeType, IKE_HEADER::Flags, IKE_PACKET::Header, IKE_HEADER_FLAGS_INIT, IKE_HEADER_FLAGS_RESPOND, IKE_NONCE_SIZE, IKE_PACKET_APPEND_PAYLOAD, IkeGenerateNonce(), IkePacketAlloc(), IkePacketFree(), IkePayloadFree(), IkeSessionTypeIkeSa, IKEV2_EXCHANGE_TYPE_INIT, IKEV2_NOTIFICATION_COOKIE, IKEV2_PAYLOAD_TYPE_KE, IKEV2_PAYLOAD_TYPE_NONCE, IKEV2_PAYLOAD_TYPE_NONE, IKEV2_PAYLOAD_TYPE_NOTIFY, IKEV2_PAYLOAD_TYPE_SA, Ikev2GenerateKePayload(), Ikev2GenerateNoncePayload(), Ikev2GenerateNotifyPayload(), Ikev2GenerateSaDhPublicKey(), Ikev2GenerateSaPayload(), IKEV2_SA_SESSION::InitiatorCookie, IKE_HEADER::InitiatorCookie, IPSEC_PROTO_ISAKMP, IKEV2_SESSION_COMMON::IsInitiator, IKE_HEADER::MessageId, IKEV2_SA_SESSION::NCookie, IKEV2_SA_SESSION::NCookieSize,

IKE_HEADER::NextPayload, IKEV2_SA_SESSION::NiBlkSize, IKEV2_SA_SESSION::NiBlock, IKEV2_SA_SESSION::NrBlkSize, IKEV2_SA_SESSION::NrBlock, IKEV2_SA_SESSION::ResponderCookie, IKE_HEADER::ResponderCookie, IKEV2_SA_SESSION::SaData, IKEV2_SA_SESSION::SessionCommon, and IKE_HEADER::Version.

Referenced by Ikev2InitCertGenerator().

EFI_STATUS Ikev2InitPskParser (IN UINT8 * *SaSession*,
IN IKE_PACKET * *IkePacket*
)

Parses the IKEv2 packet for IKE_SA_INIT exchange.

Parameters:

[in] *SaSession* Pointer to IKEV2_SA_SESSION related to the exchange.
[in] *IkePacket* The received IKE packet to be parsed.

Return values:

EFI_SUCCESS	The IKEv2 packet is acceptable and the relative data is saved for furthure communication.
EFI_INVALID_PARAMETER	The IKEv2 packet is malformed or the SA proposal is unacceptable.

References IKE_HEADER_FLAGS_INIT, IKE_HEADER_FLAGS_RESPOND, IKE_NONCE_SIZE, IKE_PAYLOAD_BY_PACKET, IkeGenerateCookie(), IkeGenerateNonce(), IkeStateAuth, IKEV2_DUMP_STATE, IKEV2_PAYLOAD_TYPE_KE, IKEV2_PAYLOAD_TYPE_NONCE, IKEV2_PAYLOAD_TYPE_NOTIFY, IKEV2_PAYLOAD_TYPE_SA, Ikev2GenerateSaDhPublicKey(), Ikev2GenerateSaKeys(), Ikev2ParserNotifyCookiePayload(), Ikev2SaParseSaPayload(), IKEV2_SA_SESSION::InitiatorCookie, IKEV2_SESSION_COMMON::IsInitiator, IKEV2_SA_SESSION::NiBlkSize, IKEV2_SA_SESSION::NiBlock, IKEV2_SA_SESSION::NrBlkSize, IKEV2_SA_SESSION::NrBlock, IKE_PAYLOAD::PayloadBuf, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, IKEV2_SESSION_COMMON::PreferDhGroup, IKEV2_SA_SESSION::ResponderCookie, IKEV2_SA_SESSION::SessionCommon, and IKEV2_SESSION_COMMON::State.

Referenced by Ikev2InitCertParser().

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED IKEV2_PACKET_HANDLER mIkev2Initial[][2]

Initial value:

```
{
 {
   {
     Ikev2InitPskParser,
     Ikev2InitPskGenerator
   },
   {
     Ikev2AuthPskParser,
     Ikev2AuthPskGenerator
   }
 },
 {
   {
     Ikev2InitCertParser,
     Ikev2InitCertGenerator
```

```
    },
    {
      Ikev2AuthCertParser,
      Ikev2AuthCertGenerator
    },
  },
}
```

Referenced by Ikev2HandleSa(), and Ikev2NegotiateSa().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/Ikev2/Utility.c File Reference

Functions

IKEV2_SA_SESSION * Ikev2SaSessionAlloc (IN IPSEC_PRIVATE_DATA *Private, IN IKE_UDP_SERVICE *UdpService)

VOID Ikev2SaSessionReg (IN IKEV2_SA_SESSION *IkeSaSession, IN IPSEC_PRIVATE_DATA *Private)

IKEV2_SA_SESSION * Ikev2SaSessionLookup (IN LIST_ENTRY *SaSessionList, IN EFI_IP_ADDRESS *RemotePeerIp)

VOID Ikev2SaSessionInsert (IN LIST_ENTRY *SaSessionList, IN IKEV2_SA_SESSION *IkeSaSession, IN EFI_IP_ADDRESS *RemotePeerIp)

IKEV2_SA_SESSION * Ikev2SaSessionRemove (IN LIST_ENTRY *SaSessionList, IN EFI_IP_ADDRESS *RemotePeerIp)

EFI_STATUS Ikev2SaSessionOnDeleting (IN IKEV2_SA_SESSION *IkeSaSession)

VOID Ikev2SaSessionCommonFree (IN IKEV2_SESSION_COMMON *SessionCommon)

VOID Ikev2SessionCommonRefresh (IN IKEV2_SESSION_COMMON *SessionCommon)

VOID Ikev2SaSessionFree (IN IKEV2_SA_SESSION *IkeSaSession)

VOID Ikev2SaSessionIncreaseMessageId (IN IKEV2_SA_SESSION *IkeSaSession)

IKEV2_CHILD_SA_SESSION * Ikev2ChildSaSessionAlloc (IN IKE_UDP_SERVICE *UdpService, IN IKEV2_SA_SESSION *IkeSaSession)

VOID Ikev2ChildSaSessionReg (IN IKEV2_CHILD_SA_SESSION *ChildSaSession, IN IPSEC_PRIVATE_DATA *Private)

IKEV2_CHILD_SA_SESSION * Ikev2ChildSaSessionLookupByMid (IN LIST_ENTRY *SaSessionList, IN UINT32 Mid)

IKEV2_CHILD_SA_SESSION * Ikev2ChildSaSessionLookupBySpi (IN LIST_ENTRY *SaSessionList, IN UINT32 Spi)

VOID Ikev2ChildSaSessionInsert (IN LIST_ENTRY *SaSessionList, IN IKEV2_CHILD_SA_SESSION *ChildSaSession)

IKEV2_CHILD_SA_SESSION * Ikev2ChildSaSessionRemove (IN LIST_ENTRY *SaSessionList, IN UINT32 Spi, IN UINT8 ListType)

EFI_STATUS Ikev2ChildSaSessionOnDeleting (IN IKEV2_CHILD_SA_SESSION *ChildSaSession)

VOID Ikev2ChildSaSessionFree (IN IKEV2_CHILD_SA_SESSION *ChildSaSession)

EFI_STATUS Ikev2ChildSaSilentDelete (IN IKEV2_SA_SESSION *IkeSaSession, IN UINT32 Spi)

VOID Ikev2DhBufferFree (IKEV2_DH_BUFFER *DhBuffer)

IKEV2_CREATE_CHILD_REQUEST_TYPE Ikev2ChildExchangeRequestType (IN IKE_PACKET *IkePacket)

EFI_STATUS Ikev2ChildSaAssociateSpdEntry (IN OUT IKEV2_CHILD_SA_SESSION *ChildSaSession)

UINT32 Ikev2ChildExchangeRekeySpi (IN IKE_PACKET *IkePacket)

BOOLEAN

Ikev2ValidateHeader (IN IKEV2_SA_SESSION *IkeSaSession, IN IKE_HEADER *IkeHdr)
IKEV2_SA_DATA * Ikev2InitializeSaData (IN IKEV2_SESSION_COMMON *SessionCommon)
VOID Ikev2StoreSaData (IN IKEV2_CHILD_SA_SESSION *ChildSaSession)
VOID EFIAPI Ikev2LifetimeNotify (IN EFI_EVENT Event, IN VOID *Context)
VOID EFIAPI Ikev2ResendNotify (IN EFI_EVENT Event, IN VOID *Context)
VOID Ikev2ChildSaSessionSpdSelectorCreate (IN OUT IKEV2_CHILD_SA_SESSION *ChildSaSession)
IKEV2_CHILD_SA_SESSION * Ikev2ChildSaSessionCreate (IN IKEV2_SA_SESSION *IkeSaSession, IN IKE_UDP_SERVICE *UdpService)
EFI_STATUS Ikev2MatchSpdEntry (IN EFI_IPSEC_CONFIG_DATA_TYPE Type, IN EFI_IPSEC_CONFIG_SELECTOR *Selector, IN VOID *Data, IN UINTN SelectorSize, IN UINTN DataSize, IN VOID *Context)
BOOLEAN Ikev2IsSupportAlg (IN UINT16 AlgorithmId, IN UINT8 Type)
VOID Ikev2ParseProposalData (IN IKEV2_PROPOSAL_DATA *ProposalData, OUT UINT16 *PreferEncryptAlgorithm, OUT UINT16 *PreferIntegrityAlgorithm, OUT UINT16 *PreferPrfAlgorithm, OUT UINT16 *PreferDhGroup, OUT UINTN *PreferEncryptKeylength, OUT BOOLEAN *IsSupportEsn, IN BOOLEAN IsChildSa)
BOOLEAN Ikev2SaParseSaPayload (IN OUT IKEV2_SA_SESSION *IkeSaSession, IN IKE_PAYLOAD *SaPayload, IN UINT8 Type)
BOOLEAN Ikev2ChildSaParseSaPayload (IN OUT IKEV2_CHILD_SA_SESSION *ChildSaSession, IN IKE_PAYLOAD *SaPayload, IN UINT8 Type)
EFI_STATUS Ikev2SaGenerateKey (IN UINT8 HashAlgId, IN UINT8 *HashKey, IN UINTN HashKeyLength, IN OUT UINT8 *OutputKey, IN UINTN OutputKeyLength, IN PRF_DATA_FRAGMENT *Fragments, IN UINTN NumFragments)

Variables

UINT16 mIkev2EncryptAlgorithmList [IKEV2_SUPPORT_ENCRYPT_ALGORITHM_NUM]
UINT16 mIkev2PrfAlgorithmList [IKEV2_SUPPORT_PRF_ALGORITHM_NUM]
UINT16 mIkev2DhGroupAlgorithmList [IKEV2_SUPPORT_DH_ALGORITHM_NUM]
UINT16 mIkev2AuthAlgorithmList [IKEV2_SUPPORT_AUTH_ALGORITHM_NUM]

Detailed Description

The Common operations used by IKE Exchange Process.

(C) Copyright 2015 Hewlett-Packard Development Company, L.P.
Copyright (c) 2010 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

UINT32 Ikev2ChildExchangeRekeySpi (IN IKE_PACKET * *IkePacket*)

This function finds the SPI from Create Child SA Exchange Packet.

Parameters:

`[in]` *IkePacket* Pointer to IKE_PACKET to be searched.

Return values:

SPI number or 0 if it is not supported.

IKEV2_CREATE_CHILD_REQUEST_TYPE Ikev2ChildExchangeRequestType (IN IKE_PACKET * *IkePacket*)

This function is to parse a request IKE packet and return its request type. The request type is one of IKE CHILD SA creation, IKE SA rekeying and IKE CHILD SA rekeying.

Parameters:

`[in]` *IkePacket* IKE packet to be prased.

return the type of the IKE packet.

References IKE_PAYLOAD_BY_PACKET, IkeRequestTypeCreateChildSa, IkeRequestTypeRekeyChildSa, IkeRequestTypeRekeyIkeSa, IKEV2_NOTIFICATION_REKEY_SA, IKEV2_PAYLOAD_TYPE_NOTIFY, IKEV2_PAYLOAD_TYPE_TS_INIT, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2HandleChildSa().

EFI_STATUS Ikev2ChildSaAssociateSpdEntry (IN OUT IKEV2_CHILD_SA_SESSION * *ChildSaSession*)

Associate a SPD selector to the Child SA Session.

This function is called when the Child SA is not the first child SA of its IKE SA. It associate a SPD to this Child SA.

Parameters:

`[in,out]` *ChildSaSession* Pointer to the Child SA Session to be associated to a SPD selector.

Return values:

EFI_SUCCESS Associate one SPD selector to this Child SA Session successfully.
EFI_NOT_FOUND Can't find the related SPD selector.

References Ikev2MatchSpdEntry(), and IpSecVisitConfigData().

Referenced by Ikev2AuthCertParser(), and Ikev2AuthPskParser().

```
BOOLEAN Ikev2ChildSaParseSaPayload (  IN OUT IKEV2_CHILD_SA_SESSION *  ChildSaSession,
                                      IN IKE_PAYLOAD *                 SaPayload,
                                      IN UINT8                         Type
                                    )
```

Parse the received Authentication Exchange Packet.

This function parse the SA Payload and Key Payload to find out the cryptographic suite for the ESP and fill it into the Child SA Session's CommonSession->SaParams.

Parameters:

`[in,out]` *ChildSaSession* Pointer to IKEV2_CHILD_SA_SESSION related to this Authentication Exchange.
`[in]` *SaPayload* The received packet.
`[in]` *Type* The IKE header's flag of received packet .

Return values:

TRUE If the SA proposal in Packet is acceptable.
FALSE If the SA proposal in Packet is not acceptable.

References IKE_HEADER_FLAGS_INIT, IKE_HEADER_FLAGS_RESPOND, Ikev2ParseProposalData(), IPSEC_PROTO_IPSEC_ESP, IKEV2_PROPOSAL_DATA::NumTransforms, IKEV2_PROPOSAL_DATA::ProtocolId, and IKEV2_PROPOSAL_DATA::Spi.

Referenced by Ikev2AuthCertParser(), and Ikev2AuthPskParser().

```
IKEV2_CHILD_SA_SESSION* Ikev2ChildSaSessionAlloc (  IN IKE_UDP_SERVICE *    UdpService,
                                                    IN IKEV2_SA_SESSION *   IkeSaSession
                                                  )
```

Allocate memory for IKEV2 Child SA Session.

Parameters:

`[in]` *UdpService* Pointer to IKE_UDP_SERVICE.
`[in]` *IkeSaSession* Pointer to IKEV2_SA_SESSION related to this Child SA Session.

Return values:

Pointer of a new created IKEV2 Child SA Session or NULL.

References IKEV2_SESSION_COMMON::AfterEncodePayload, IKEV2_SESSION_COMMON::BeforeDecodePayload, IkeGenerateSpi(), IKEV2_CHILD_SA_SESSION::IkeSaSession, IKEV2_SESSION_COMMON::IkeSessionType, IkeSessionTypeChildSa, IKEV2_CHILD_SA_SESSION_SIGNATURE,

Ikev2ChildSaAfterEncodePayload(), Ikev2ChildSaBeforeDecodePayload(), Ikev2ResendNotify(), IKEV2_SESSION_COMMON::IkeVer, IKEV2_SESSION_COMMON::LocalPeerIp, IKEV2_CHILD_SA_SESSION::LocalPeerSpi, IKEV2_CHILD_SA_SESSION::MessageId, IKEV2_SESSION_COMMON::Private, IKEV2_SESSION_COMMON::RemotePeerIp, IKEV2_SA_SESSION::SessionCommon, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_CHILD_SA_SESSION::Signature, IKEV2_SESSION_COMMON::TimeoutEvent, and IKEV2_SESSION_COMMON::UdpService.

Referenced by Ikev2ChildSaSessionCreate(), and Ikev2NegotiateChildSa().

IKEV2_CHILD_SA_SESSION*
Ikev2ChildSaSessionCreate (IN IKEV2_SA_SESSION * *IkeSaSession*,
IN IKE_UDP_SERVICE * *UdpService*
)

Generate a ChildSa Session and insert it into related IkeSaSession.

Parameters:

[in] *IkeSaSession* Pointer to related IKEV2_SA_SESSION.
[in] *UdpService* Pointer to related IKE_UDP_SERVICE.

Returns:

pointer of IKEV2_CHILD_SA_SESSION.

References IkeStateAuth, IkeStateCreateChild, IKEV2_DUMP_STATE, Ikev2ChildSaSessionAlloc(), Ikev2ChildSaSessionInsert(), Ikev2ChildSaSessionSpdSelectorCreate(), IKEV2_SESSION_COMMON::IsInitiator, IKEV2_CHILD_SA_SESSION::LocalPort, IKEV2_CHILD_SA_SESSION::NiBlkSize, IKEV2_CHILD_SA_SESSION::NiBlock, IKEV2_CHILD_SA_SESSION::NrBlkSize, IKEV2_CHILD_SA_SESSION::NrBlock, IKEV2_CHILD_SA_SESSION::ProtoId, IKEV2_CHILD_SA_SESSION::RemotePort, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_CHILD_SA_SESSION::Spd, and IKEV2_SESSION_COMMON::State.

Referenced by Ikev2HandleSa().

VOID Ikev2ChildSaSessionFree (IN IKEV2_CHILD_SA_SESSION * *ChildSaSession*)

Free the memory located for the specified IKEV2_CHILD_SA_SESSION.

Parameters:

[in] *ChildSaSession* Pointer to IKEV2_CHILD_SA_SESSION.

References Ikev2DhBufferFree(), and Ikev2SaSessionCommonFree().

Referenced by Ikev2ChildSaSessionReg(), Ikev2ChildSaSilentDelete(), Ikev2HandleSa(), Ikev2NegotiateChildSa(), Ikev2ResendNotify(), and Ikev2SaSessionFree().

VOID Ikev2ChildSaSessionInsert (IN LIST_ENTRY * *SaSessionList*,
IN IKEV2_CHILD_SA_SESSION * *ChildSaSession*
)

Insert a Child SA Session into the specified ChildSa list.

Parameters:

[in] *SaSessionList* Pointer to list to be inserted in.
[in] *ChildSaSession* Pointer to IKEV2_CHILD_SA_SESSION to be inserted.

Referenced by Ikev2ChildSaSessionCreate(), Ikev2ChildSaSessionReg(), and Ikev2NegotiateChildSa().

IKEV2_CHILD_SA_SESSION*
Ikev2ChildSaSessionLookupByMid (IN LIST_ENTRY * *SaSessionList*,
IN UINT32 *Mid*
)

Find the ChildSaSession by it's MessagId.

Parameters:

[in] *SaSessionList* Pointer to a ChildSaSession List.
[in] *Mid* The messageId used to search ChildSaSession.

Returns:

Pointer to IKEV2_CHILD_SA_SESSION or NULL.

References IKEV2_CHILD_SA_SESSION_BY_IKE_SA, and IKEV2_CHILD_SA_SESSION::MessageId.

IKEV2_CHILD_SA_SESSION*
Ikev2ChildSaSessionLookupBySpi (IN LIST_ENTRY * *SaSessionList*,
IN UINT32 *Spi*
)

This function find the Child SA by the specified SPI.

This functin find a ChildSA session by searching the ChildSaSessionlist of the input IKEV2_SA_SESSION by specified MessageID.

Parameters:

[in] *SaSessionList* Pointer to List to be searched.
[in] *Spi* Specified SPI.

Returns:

Pointer to IKEV2_CHILD_SA_SESSION or NULL.

References IKEV2_CHILD_SA_SESSION_BY_IKE_SA, IKEV2_CHILD_SA_SESSION::LocalPeerSpi, and IKEV2_CHILD_SA_SESSION::RemotePeerSpi.

Referenced by Ikev2OnPacketSent().

EFI_STATUS Ikev2ChildSaSessionOnDeleting (IN IKEV2_CHILD_SA_SESSION * *ChildSaSession*)

Mark a specified Child SA Session as on deleting.

Parameters:

[in] *ChildSaSession* Pointer to IKEV2_CHILD_SA_SESSION.

Return values:

EFI_SUCCESS Operation is successful.

```
VOID Ikev2ChildSaSessionReg ( IN IKEV2_CHILD_SA_SESSION *   ChildSaSession,
                              IN IPSEC_PRIVATE_DATA *       Private
                            )
```

Register a established IKEv2 Child SA into IkeSaSession->ChildSaEstablishSessionList. If the there is IKEV2_CHILD_SA_SESSION with same remote peer IP, remove the old one then register the new one.

Parameters:

[in]	*ChildSaSession*	Pointer to IKEV2_CHILD_SA_SESSION to be registered.
[in]	*Private*	Pointer to IPSEC_PRAVATE_DATA.

References CHILD_SA_DEFAULT_LIFETIME, IKEV2_SA_SESSION::ChildSaEstablishSessionList, IKEV2_ESTABLISHED_CHILDSA_LIST, Ikev2ChildSaSessionFree(), Ikev2ChildSaSessionInsert(), Ikev2ChildSaSessionRemove(), Ikev2LifetimeNotify(), Ikev2SessionCommonRefresh(), Ikev2StoreSaData(), and IKEV2_SESSION_COMMON::TimeoutEvent.

Referenced by Ikev2HandleSa().

```
IKEV2_CHILD_SA_SESSION* Ikev2ChildSaSessionRemove ( IN LIST_ENTRY *   SaSessionList,
                                                    IN UINT32         Spi,
                                                    IN UINT8          ListType
                                                  )
```

Remove the IKEV2_CHILD_SA_SESSION from IkeSaSessionList.

Parameters:

[in]	*SaSessionList*	The SA Session List to be iterated.
[in]	*Spi*	Spi used to identified the IKEV2_CHILD_SA_SESSION.
[in]	*ListType*	The type of the List to indicate whether it is a Established.

Returns:

The point to IKEV2_CHILD_SA_SESSION or NULL.

References IKEV2_CHILD_SA_SESSION_BY_DEL_SA, IKEV2_CHILD_SA_SESSION_BY_IKE_SA, IKEV2_DELET_CHILDSA_LIST, IKEV2_ESTABLISHED_CHILDSA_LIST, IKEV2_ESTABLISHING_CHILDSA_LIST, IKEV2_CHILD_SA_SESSION::LocalPeerSpi, and IKEV2_CHILD_SA_SESSION::RemotePeerSpi.

Referenced by Ikev2ChildSaSessionReg(), Ikev2ChildSaSilentDelete(), Ikev2HandleSa(), Ikev2OnPacketSent(), and Ikev2ResendNotify().

```
VOID Ikev2ChildSaSessionSpdSelectorCreate ( IN OUT IKEV2_CHILD_SA_SESSION *   ChildSaSession )
```

Copy ChildSaSession->Spd->Selector to ChildSaSession->SpdSelector.

ChildSaSession->SpdSelector stores the real Spdselector for its SA. Sometime, the SpdSelector in ChildSaSession is more accurated or the scope is smaller than the one in ChildSaSession->Spd, especially for the tunnel mode.

Parameters:

[in,out]	*ChildSaSession*	Pointer to IKEV2_CHILD_SA_SESSION related to.

Referenced by Ikev2AuthCertParser(), Ikev2AuthPskParser(), and Ikev2ChildSaSessionCreate().

EFI_STATUS Ikev2ChildSaSilentDelete (IN IKEV2_SA_SESSION * *IkeSaSession*,
IN UINT32 *Spi*
)

Delete the specified established Child SA.

This function delete the Child SA directly and don't send the Information Packet to remote peer.

Parameters:

[in] *IkeSaSession* Pointer to a IKE SA Session used to be searched for.
[in] *Spi* SPI used to find the Child SA.

Return values:

EFI_NOT_FOUND Pointer of IKE SA Session is NULL.
EFI_NOT_FOUND There is no specified Child SA related with the input SPI under this IKE SA Session.
EFI_SUCCESS Delete the Child SA successfully.

References EfiIpSecConfigGetNextSelector(), EfiIpSecConfigSetData(), IKEV2_ESTABLISHED_CHILDSA_LIST, Ikev2ChildSaSessionFree(), Ikev2ChildSaSessionRemove(), _IPSEC_PRIVATE_DATA::IpSecConfig, IKEV2_CHILD_SA_SESSION::LocalPeerSpi, and IKEV2_CHILD_SA_SESSION::RemotePeerSpi.

Referenced by Ikev2InfoParser(), Ikev2OnPacketSent(), Ikev2ResendNotify(), and Ikev2SaSessionFree().

VOID Ikev2DhBufferFree (IKEV2_DH_BUFFER * *DhBuffer*)

Free the specified DhBuffer.

Parameters:

[in] *DhBuffer* Pointer to IKEV2_DH_BUFFER to be freed.

References IKEV2_DH_BUFFER::DhContext, IKEV2_DH_BUFFER::GxBuffer, IKEV2_DH_BUFFER::GxyBuffer, IKEV2_DH_BUFFER::GyBuffer, and IpSecCryptoIoFreeDh().

IKEV2_SA_DATA* Ikev2InitializeSaData (IN IKEV2_SESSION_COMMON * *SessionCommon*)

Create and intialize IKEV2_SA_DATA for speicifed IKEV2_SESSION_COMMON.

This function will be only called by the initiator. The responder's IKEV2_SA_DATA will be generated during parsed the initiator packet.

Parameters:

[in] *SessionCommon* Pointer to IKEV2_SESSION_COMMON related to.

Return values:

a Pointer to a new IKEV2_SA_DATA or NULL.

References IKE_SA_ATTRIBUTE::Attr, IKEV2_TRANSFORM_DATA::Attribute, IKE_SA_ATTR_UNION::AttrLength, IKE_SA_ATTRIBUTE::AttrType, IkeSessionTypeIkeSa, IKEV2_ATTRIBUTE_TYPE_KEYLEN, IKEV2_CHILD_SA_SESSION_FROM_COMMON,

IKEV2_TRANSFORM_ID_AUTH_HMAC_SHA1_96, IKEV2_TRANSFORM_ID_DH_1024MODP, IKEV2_TRANSFORM_ID_ENCR_3DES, IKEV2_TRANSFORM_ID_ENCR_AES_CBC, IKEV2_TRANSFORM_ID_PRF_HMAC_SHA1, IKEV2_TRANSFORM_TYPE_DH, IKEV2_TRANSFORM_TYPE_ENCR, IKEV2_TRANSFORM_TYPE_ESN, IKEV2_TRANSFORM_TYPE_INTEG, IKEV2_TRANSFORM_TYPE_PRF, IPSEC_PROTO_IPSEC_ESP, IPSEC_PROTO_ISAKMP, IpSecGetEncryptKeyLength(), IKEV2_CHILD_SA_SESSION::LocalPeerSpi, IKEV2_SA_DATA::NumProposals, IKEV2_PROPOSAL_DATA::NumTransforms, IKEV2_PROPOSAL_DATA::ProposalIndex, IKEV2_PROPOSAL_DATA::ProtocolId, IKEV2_PROPOSAL_DATA::Spi, IKEV2_TRANSFORM_DATA::TransformId, IKEV2_TRANSFORM_DATA::TransformIndex, and IKEV2_TRANSFORM_DATA::TransformType.

Referenced by Ikev2HandleSa(), Ikev2NegotiateChildSa(), and Ikev2NegotiateSa().

```
BOOLEAN Ikev2IsSupportAlg ( IN UINT16  AlgorithmId,
                            IN UINT8   Type
                          )
```

Check if the Algorithm ID is supported.

Parameters:

[in]	*AlgorithmId*	The specified Algorithm ID.
[in]	*Type*	The type used to indicate the Algorithm is for Encrypt or Authentication.

Return values:

TRUE	If the Algorithm ID is supported.
FALSE	If the Algorithm ID is not supported.

References IKE_AUTH_TYPE, IKE_DH_TYPE, IKE_ENCRYPT_TYPE, IKE_PRF_TYPE, IKEV2_SUPPORT_AUTH_ALGORITHM_NUM, IKEV2_SUPPORT_DH_ALGORITHM_NUM, IKEV2_SUPPORT_ENCRYPT_ALGORITHM_NUM, IKEV2_SUPPORT_PRF_ALGORITHM_NUM, mIkev2AuthAlgorithmList, mIkev2DhGroupAlgorithmList, mIkev2EncryptAlgorithmList, and mIkev2PrfAlgorithmList.

Referenced by Ikev2ParseProposalData().

```
VOID EFIAPI Ikev2LifetimeNotify ( IN EFI_EVENT  Event,
                                  IN VOID *    Context
                                )
```

Call back function of the IKE life time is over.

This function will mark the related IKE SA Session as deleting and trigger a Information negotiation.

Parameters:

[in]	*Event*	The signaled Event.
[in]	*Context*	Pointer to data passed by caller.

References IKEV2_CHILD_SA_SESSION::ByDelete, IKEV2_SA_SESSION::DeleteSaList, IKEV2_CHILD_SA_SESSION::IkeSaSession, IKEV2_SESSION_COMMON::IkeSessionType, IkeSessionTypeIkeSa, IkeStateSaDeleting, IKEV2_CHILD_SA_SESSION_FROM_COMMON, IKEV2_DUMP_STATE, IKEV2_SA_SESSION_FROM_COMMON, IKEV2_SA_SESSION::InitiatorCookie, IKEV2_CHILD_SA_SESSION::LocalPeerSpi, mIkev2Exchange, _IKE_EXCHANGE_INTERFACE::NegotiateInfo, IKEV2_SA_SESSION::ResponderCookie,

IKEV2_SA_SESSION::SessionCommon, and IKEV2_SESSION_COMMON::State.

Referenced by Ikev2ChildSaSessionReg(), and Ikev2SaSessionReg().

```
EFI_STATUS Ikev2MatchSpdEntry ( IN EFI_IPSEC_CONFIG_DATA_TYPE   Type,
                                IN EFI_IPSEC_CONFIG_SELECTOR *  Selector,
                                IN VOID *                       Data,
                                IN UINTN                        SelectorSize,
                                IN UINTN                        DataSize,
                                IN VOID *                       Context
                              )
```

Check if the SPD is related to the input Child SA Session.

This function is the subfunction of Ikev1AssociateSpdEntry(). It is the call back function of IpSecVisitConfigData().

Parameters:

[in]	*Type*	Type of the input Config Selector.
[in]	*Selector*	Pointer to the Configure Selector to be checked.
[in]	*Data*	Pointer to the Configure Selector's Data passed from the caller.
[in]	*SelectorSize*	The buffer size of Selector.
[in]	*DataSize*	The buffer size of the Data.
[in]	*Context*	The data passed from the caller. It is a Child SA Session in this context.

Return values:

EFI_SUCCESS	The SPD Selector is not related to the Child SA Session.
EFI_ABORTED	The SPD Selector is related to the Child SA session and set the ChildSaSession->Spd to point to this SPD Selector.

References EFI_IPSEC_ANY_PORT, EFI_IPSEC_ANY_PROTOCOL, IKE_DEFAULT_PORT, IkeSearchSpdEntry(), IpSecMatchIpAddress(), IKE_UDP_SERVICE::IpVersion, IKEV2_SESSION_COMMON::LocalPeerIp, IKEV2_CHILD_SA_SESSION::LocalPort, IKEV2_CHILD_SA_SESSION::ProtoId, IKEV2_SESSION_COMMON::RemotePeerIp, IKEV2_CHILD_SA_SESSION::RemotePort, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_CHILD_SA_SESSION::Spd, and IKEV2_SESSION_COMMON::UdpService.

Referenced by Ikev2ChildSaAssociateSpdEntry().

```
VOID Ikev2ParseProposalData ( IN IKEV2_PROPOSAL_DATA *   ProposalData,
                              OUT UINT16 *               PreferEncryptAlgorithm,
                              OUT UINT16 *               PreferIntegrityAlgorithm,
                              OUT UINT16 *               PreferPrfAlgorithm,
                              OUT UINT16 *               PreferDhGroup,
                              OUT UINTN *                PreferEncryptKeylength,
                              OUT BOOLEAN *              IsSupportEsn,
                              IN BOOLEAN                 IsChildSa
                            )
```

Get the preferred algorithm types from ProposalData.

Parameters:

[in]	*ProposalData*	Pointer to related IKEV2_PROPOSAL_DATA.
[out]	*PreferEncryptAlgorithm*	Output of preferred encrypt algorithm.
[out]	*PreferIntegrityAlgorithm*	Output of preferred integrity algorithm.
[out]	*PreferPrfAlgorithm*	Output of preferred PRF algorithm. Only for IKE SA.
[out]	*PreferDhGroup*	Output of preferred DH group. Only for IKE SA.
[out]	*PreferEncryptKeylength*	Output of preferred encrypt key length in bytes.
[out]	*IsSupportEsn*	Output of value about the Extented Sequence Number is support or not. Only for Child SA.
[in]	*IsChildSa*	If it is ture, the ProposalData is for IKE SA. Otherwise the proposalData is for Child SA.

References IKE_SA_ATTRIBUTE::Attr, IKEV2_TRANSFORM_DATA::Attribute, IKE_SA_ATTRIBUTE::AttrType, IKE_SA_ATTR_UNION::AttrValue, IKE_AUTH_TYPE, IKE_DH_TYPE, IKE_ENCRYPT_TYPE, IKE_PRF_TYPE, IKEV2_ATTRIBUTE_TYPE_KEYLEN, IKEV2_TRANSFORM_TYPE_DH, IKEV2_TRANSFORM_TYPE_ENCR, IKEV2_TRANSFORM_TYPE_ESN, IKEV2_TRANSFORM_TYPE_INTEG, IKEV2_TRANSFORM_TYPE_PRF, Ikev2IsSupportAlg(), IpSecGetEncryptKeyLength(), IKEV2_TRANSFORM_DATA::TransformId, and IKEV2_TRANSFORM_DATA::TransformType.

Referenced by Ikev2ChildSaParseSaPayload(), and Ikev2SaParseSaPayload().

```
VOID EFIAPI Ikev2ResendNotify ( IN EFI_EVENT  Event,
                                IN VOID *     Context
                              )
```

This function will be called if the TimeOut Event is signaled.

Parameters:

[in]	*Event*	The signaled Event.
[in]	*Context*	The data passed by caller.

References IKEV2_SA_SESSION::ChildSaEstablishSessionList, IKEV2_SA_SESSION::ChildSaSessionList, IKEV2_SA_SESSION::DeleteSaList, IKE_MAX_RETRY, IKEV2_CHILD_SA_SESSION::IkeSaSession, IKEV2_SESSION_COMMON::IkeSessionType, IkeSessionTypeIkeSa, IkeStateSaDeleting, _IPSEC_PRIVATE_DATA::Ikev1EstablishedList, IKEV2_CHILD_SA_SESSION_BY_IKE_SA, IKEV2_CHILD_SA_SESSION_FROM_COMMON, IKEV2_DELET_CHILDSA_LIST, IKEV2_ESTABLISHING_CHILDSA_LIST, IKEV2_SA_SESSION_FROM_COMMON, Ikev2ChildSaSessionFree(), Ikev2ChildSaSessionRemove(), Ikev2ChildSaSilentDelete(), _IPSEC_PRIVATE_DATA::Ikev2EstablishedList, Ikev2SaSessionFree(), Ikev2SaSessionRemove(), Ikev2SendIkePacket(), _IPSEC_PRIVATE_DATA::Ikev2SessionList, _IPSEC_PRIVATE_DATA::IpSec, IPSEC_STATUS_DISABLED, IPSECCONFIG_STATUS_NAME, _IPSEC_PRIVATE_DATA::IsIPsecDisabling, IKEV2_SESSION_COMMON::LastSentPacket, IKEV2_CHILD_SA_SESSION::LocalPeerSpi, IKEV2_SESSION_COMMON::Private, IKEV2_SESSION_COMMON::RemotePeerIp, IKEV2_SESSION_COMMON::RetryCount, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_SA_SESSION::SessionCommon, IKEV2_SESSION_COMMON::State, and IKEV2_SESSION_COMMON::UdpService.

Referenced by Ikev2ChildSaSessionAlloc(), and Ikev2SaSessionAlloc().

```
EFI_STATUS Ikev2SaGenerateKey ( IN UINT8        HashAlgId,
                                IN UINT8 *      HashKey,
```

```
                                IN UINTN                    HashKeyLength,
                                IN OUT UINT8 *              OutputKey,
                                IN UINTN                    OutputKeyLength,
                                IN PRF_DATA_FRAGMENT *      Fragments,
                                IN UINTN                    NumFragments
                              )
```

Generate Key buffer from fragments.

If the digest length of specified HashAlgId is larger than or equal with the required output key length, derive the key directly. Otherwise, Key Material needs to be PRF-based concatenation according to 2.13 of RFC 4306: prf+ (K,S) = T1 | T2 | T3 | T4 | ..., T1 = prf (K, S | 0x01), T2 = prf (K, T1 | S | 0x02), T3 = prf (K, T2 | S | 0x03),T4 = prf (K, T3 | S | 0x04) then derive the key from this key material.

Parameters:

[in]	*HashAlgId*	The Hash Algorithm ID used to generate key.
[in]	*HashKey*	Pointer to a key buffer which contains hash key.
[in]	*HashKeyLength*	The length of HashKey in bytes.
[in,out]	*OutputKey*	Pointer to buffer which is used to receive the output key.
[in]	*OutputKeyLength*	The length of OutPutKey buffer.
[in]	*Fragments*	Pointer to the data to be used to generate key.
[in]	*NumFragments*	The numbers of the Fragement.

Return values:

EFI_SUCCESS	The operation complete successfully.
EFI_INVALID_PARAMETER	If NumFragments is zero.
EFI_OUT_OF_RESOURCES	If the required resource can't be allocated.
Others	The operation is failed.

References PRF_DATA_FRAGMENT::Data, PRF_DATA_FRAGMENT::DataSize, IpSecCryptoIoHmac(), and IpSecGetHmacDigestLength().

Referenced by Ikev2GenerateChildSaKeys(), and Ikev2GenerateSaKeys().

```
BOOLEAN Ikev2SaParseSaPayload ( IN OUT IKEV2_SA_SESSION *   IkeSaSession,
                                IN IKE_PAYLOAD *            SaPayload,
                                IN UINT8                    Type
                              )
```

Parse the received Initial Exchange Packet.

This function parse the SA Payload and Key Payload to find out the cryptographic suite for the further IKE negotiation and fill it into the IKE SA Session's CommonSession->SaParams.

Parameters:

[in,out]	*IkeSaSession*	Pointer to related IKEV2_SA_SESSION.
[in]	*SaPayload*	The received packet.
[in]	*Type*	The received packet IKE header flag.

Return values:

TRUE	If the SA proposal in Packet is acceptable.
FALSE	If the SA proposal in Packet is not acceptable.

References IKE_HEADER_FLAGS_INIT, IKE_HEADER_FLAGS_RESPOND, Ikev2ParseProposalData(), IPSEC_PROTO_ISAKMP, IKEV2_PROPOSAL_DATA::NumTransforms, and IKEV2_PROPOSAL_DATA::ProtocolId.

Referenced by Ikev2InitPskParser().

IKEV2_SA_SESSION* Ikev2SaSessionAlloc (IN IPSEC_PRIVATE_DATA * *Private*,
IN IKE_UDP_SERVICE * *UdpService*
)

Allocate buffer for IKEV2_SA_SESSION and initialize it.

Parameters:

[in] *Private* Pointer to IPSEC_PRIVATE_DATA.
[in] *UdpService* Pointer to IKE_UDP_SERVICE related to this IKE SA Session.

Returns:

Pointer to IKEV2_SA_SESSION or NULL.

References IKEV2_SESSION_COMMON::AfterEncodePayload, IKEV2_SESSION_COMMON::BeforeDecodePayload, IKEV2_SA_SESSION::ChildSaEstablishSessionList, IKEV2_SA_SESSION::ChildSaSessionList, IKEV2_SA_SESSION::DeleteSaList, IkeGenerateCookie(), IKEV2_SESSION_COMMON::IkeSessionType, IkeSessionTypeIkeSa, IKEV2_SA_SESSION_SIGNATURE, Ikev2ResendNotify(), IKEV2_SESSION_COMMON::IkeVer, IKEV2_SA_SESSION::InfoMIDList, IKEV2_SA_SESSION::InitiatorCookie, IKEV2_SA_SESSION::MessageId, IKEV2_SA_SESSION::NCookie, IKEV2_SESSION_COMMON::Private, IKEV2_SA_SESSION::ResponderCookie, IKEV2_SA_SESSION::SessionCommon, IKEV2_SA_SESSION::Signature, IKEV2_SESSION_COMMON::TimeoutEvent, and IKEV2_SESSION_COMMON::UdpService.

Referenced by Ikev2HandleSa(), and Ikev2NegotiateSa().

VOID Ikev2SaSessionCommonFree (IN IKEV2_SESSION_COMMON * *SessionCommon*)

Free specified Seession Common. The session common would belong to a IKE SA or a Child SA.

Parameters:

[in] *SessionCommon* Pointer to a Session Common.

References IkePacketFree().

Referenced by Ikev2ChildSaSessionFree(), and Ikev2SaSessionFree().

VOID Ikev2SaSessionFree (IN IKEV2_SA_SESSION * *IkeSaSession*)

Free specified IKEV2 SA Session.

Parameters:

[in] *IkeSaSession* Pointer to IKEV2_SA_SESSION to be freed.

References IKEV2_SESSION_KEYS::DhBuffer, IKEV2_CHILD_SA_SESSION::IkeSaSession, IKEV2_CHILD_SA_SESSION_BY_IKE_SA, Ikev2ChildSaSessionFree(), Ikev2ChildSaSilentDelete(), Ikev2DhBufferFree(), Ikev2SaSessionCommonFree(), IKEV2_CHILD_SA_SESSION::LocalPeerSpi,

IKEV2_SESSION_KEYS::SkAiKey, IKEV2_SESSION_KEYS::SkArKey, IKEV2_SESSION_KEYS::SkdKey, IKEV2_SESSION_KEYS::SkEiKey, IKEV2_SESSION_KEYS::SkErKey, IKEV2_SESSION_KEYS::SkPiKey, and IKEV2_SESSION_KEYS::SkPrKey.

Referenced by IkeDeleteAllSas(), Ikev2HandleSa(), Ikev2InfoParser(), Ikev2NegotiateSa(), Ikev2OnPacketSent(), Ikev2ResendNotify(), Ikev2SaSessionReg(), and IpSecStop().

```
VOID Ikev2SaSessionIncreaseMessageId ( IN IKEV2_SA_SESSION * IkeSaSession )
```

Increase the MessageID in IkeSaSession.

Parameters:

[in] *IkeSaSession* Pointer to a specified IKEV2_SA_SESSION.

Referenced by Ikev2InfoGenerator().

```
VOID Ikev2SaSessionInsert ( IN LIST_ENTRY *          SaSessionList,
                            IN IKEV2_SA_SESSION *    IkeSaSession,
                            IN EFI_IP_ADDRESS *      RemotePeerIp
                          )
```

Insert a IKE_SA_SESSION into IkeSaSession list. The IkeSaSession list is either Private->Ikev2SaSession list or Private->Ikev2EstablishedList list.

Parameters:

[in] *SaSessionList* Pointer to list to be inserted into.
[in] *IkeSaSession* Pointer to IKEV2_SA_SESSION to be inserted.
[in] *RemotePeerIp* Pointer to EFI_IP_ADDRESSS to indicate the unique IKEV2_SA_SESSION.

References Ikev2SaSessionRemove().

Referenced by Ikev2HandleSa(), Ikev2NegotiateSa(), and Ikev2SaSessionReg().

```
IKEV2_SA_SESSION* Ikev2SaSessionLookup ( IN LIST_ENTRY *        SaSessionList,
                                         IN EFI_IP_ADDRESS *    RemotePeerIp
                                       )
```

Find a IKEV2_SA_SESSION by the remote peer IP.

Parameters:

[in] *SaSessionList* SaSession List to be searched.
[in] *RemotePeerIp* Pointer to specified IP address.

Returns:

Pointer to IKEV2_SA_SESSION if find one or NULL.

References IKEV2_SA_SESSION_BY_SESSION, IKEV2_SESSION_COMMON::RemotePeerIp, and IKEV2_SA_SESSION::SessionCommon.

Referenced by IkeNegotiate(), Ikev2HandleChildSa(), Ikev2HandleInfo(), Ikev2HandleSa(), Ikev2NegotiateSa(), and Ikev2OnPacketSent().

EFI_STATUS Ikev2SaSessionOnDeleting (IN IKEV2_SA_SESSION * *IkeSaSession*)

Marking a SA session as on deleting.

Parameters:

[in] *IkeSaSession* Pointer to IKEV2_SA_SESSION.

Return values:

EFI_SUCCESS Find the related SA session and marked it.

VOID Ikev2SaSessionReg (IN IKEV2_SA_SESSION * *IkeSaSession*,
IN IPSEC_PRIVATE_DATA * *Private*
)

Register the established IKEv2 SA into Private->Ikev2EstablishedList. If there is IKEV2_SA_SESSION with same remote peer IP, remove the old one then register the new one.

Parameters:

[in] *IkeSaSession* Pointer to IKEV2_SA_SESSION to be registered.

[in] *Private* Pointer to IPSEC_PRAVATE_DATA.

References IKE_SA_DEFAULT_LIFETIME, Ikev2LifetimeNotify(), Ikev2SaSessionFree(), Ikev2SaSessionInsert(), Ikev2SaSessionRemove(), Ikev2SessionCommonRefresh(), IKEV2_SESSION_COMMON::RemotePeerIp, and IKEV2_SESSION_COMMON::TimeoutEvent.

Referenced by Ikev2HandleSa().

IKEV2_SA_SESSION* Ikev2SaSessionRemove (IN LIST_ENTRY * *SaSessionList*,
IN EFI_IP_ADDRESS * *RemotePeerIp*
)

Remove the SA Session by Remote Peer IP.

Parameters:

[in] *SaSessionList* Pointer to list to be searched.

[in] *RemotePeerIp* Pointer to EFI_IP_ADDRESS to use for SA Session search.

Return values:

Pointer to IKEV2_SA_SESSION with the specified remote IP address or NULL.

References IKEV2_SA_SESSION_BY_SESSION, IKEV2_SESSION_COMMON::RemotePeerIp, and IKEV2_SA_SESSION::SessionCommon.

Referenced by Ikev2HandleSa(), Ikev2ResendNotify(), Ikev2SaSessionInsert(), and Ikev2SaSessionReg().

VOID Ikev2SessionCommonRefresh (IN IKEV2_SESSION_COMMON * *SessionCommon*)

After IKE/Child SA is estiblished, close the time event and free sent packet.

Parameters:

[in] *SessionCommon* Pointer to a Session Common.

References IkePacketFree().

Referenced by Ikev2ChildSaSessionReg(), and Ikev2SaSessionReg().

VOID Ikev2StoreSaData (IN IKEV2_CHILD_SA_SESSION * *ChildSaSession*)
Store the SA into SAD.

Parameters:

[in] *ChildSaSession* Pointer to IKEV2_CHILD_SA_SESSION.

References EfiIpSecConfigSetData(), _IPSEC_PRIVATE_DATA::IpSecConfig, and IKEV2_SESSION_COMMON::Private.

Referenced by Ikev2ChildSaSessionReg().

BOOLEAN Ikev2ValidateHeader (IN IKEV2_SA_SESSION * *IkeSaSession*,
IN IKE_HEADER * *IkeHdr*
)
Validate the IKE header of received IKE packet.

Parameters:

[in] *IkeSaSession* Pointer to IKEV2_SA_SESSION related to this IKE packet.
[in] *IkeHdr* Pointer to IKE header of received IKE packet.

Return values:

TRUE If the IKE header is valid.
FALSE If the IKE header is invalid.

References IKE_HEADER_FLAGS_INIT, IKE_HEADER_FLAGS_RESPOND, IkeStateAuth, IkeStateInit, IKEV2_EXCHANGE_TYPE_CREATE_CHILD, and IKEV2_EXCHANGE_TYPE_INFO.

Referenced by Ikev2HandleChildSa(), Ikev2HandleInfo(), and Ikev2HandleSa().

Variable Documentation

UINT16 mIkev2AuthAlgorithmList[IKEV2_SUPPORT_AUTH_ALGORITHM_NUM]
Initial value:

```
{
 IKEV2_TRANSFORM_ID_AUTH_HMAC_SHA1_96,
}
```

Referenced by Ikev2IsSupportAlg().

UINT16 mIkev2DhGroupAlgorithmList[IKEV2_SUPPORT_DH_ALGORITHM_NUM]
Initial value:

```
{
 IKEV2_TRANSFORM_ID_DH_1024MODP,
 IKEV2_TRANSFORM_ID_DH_2048MODP,
```

```
}
```

Referenced by Ikev2IsSupportAlg().

UINT16 mIkev2EncryptAlgorithmList[IKEV2_SUPPORT_ENCRYPT_ALGORITHM_NUM]

Initial value:

```
{
  IKEV2_TRANSFORM_ID_ENCR_3DES,
  IKEV2_TRANSFORM_ID_ENCR_AES_CBC,
}
```

Referenced by Ikev2IsSupportAlg().

UINT16 mIkev2PrfAlgorithmList[IKEV2_SUPPORT_PRF_ALGORITHM_NUM]

Initial value:

```
{
  IKEV2_TRANSFORM_ID_PRF_HMAC_SHA1,
}
```

Referenced by Ikev2IsSupportAlg().

Generated on Mon Sep 28 08:49:54 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IpSecDxe/Ikev2/Utility.h File Reference

Defines

"memItemRight" valign="bottom">IKEV2_SUPPORT_ENCRYPT_ALGORITHM_NUM 2

#define IKEV2_SUPPORT_PR

#define IKEV2_SUPPORT_DH

#define IKEV2_SUPPORT_AU

Functions

IKEV2_SA_SESSION * Ikev2SaSessionAlloc (I
*Private, IN IKE_UDP_
VOID Ikev2SaSessionReg (IN
*IkeSaSession, IN IPSE
*Private)
IKEV2_SA_SESSION * Ikev2SaSessionLookup
*SaSessionList, IN EFI
*RemotePeerIp)
VOID Ikev2SaSessionInsert (I
*SaSessionList, IN IKE
*IkeSaSession, IN EFI_
*RemotePeerIp)
IKEV2_SA_SESSION * Ikev2SaSessionRemove
*SaSessionList, IN EFI
*RemotePeerIp)
EFI_STATUS Ikev2SaSessionOnDele
IKEV2_SA_SESSION
VOID Ikev2SessionCommonF
IKEV2_SESSION_CO
VOID Ikev2SaSessionFree (IN
*IkeSaSession)
VOID Ikev2SaSessionCommo
IKEV2_SESSION_CO
VOID Ikev2SaSessionIncrease
IKEV2_SA_SESSION
IKEV2_CHILD_SA_SESSION * Ikev2ChildSaSessionA
*UdpService, IN IKEV
*IkeSaSession)
VOID Ikev2ChildSaSessionRe
IKEV2_CHILD_SA_S
IPSEC_PRIVATE_DA
IKEV2_CHILD_SA_SESSION * Ikev2ChildSaSessionLo
LIST_ENTRY *SaSess
IKEV2_CHILD_SA_SESSION * Ikev2ChildSaSessionLo
LIST_ENTRY *SaSess
VOID Ikev2ChildSaSessionIn
*SaSessionList, IN IKE
*ChildSaSession)
IKEV2_CHILD_SA_SESSION *

Ikev2ChildSaSessionRemove (I
*SaSessionList, IN UINT32 Sp
EFI_STATUS Ikev2ChildSaSessionOnDeletin
IKEV2_CHILD_SA_SESSION
VOID Ikev2ChildSaSessionFree (IN
IKEV2_CHILD_SA_SESSION
VOID Ikev2DhBufferFree (IN IKEV2
*DhBuffer)
EFI_STATUS Ikev2ChildSaSilentDelete (IN I
*IkeSaSession, IN UINT32 Spi
IKEV2_CREATE_CHILD_REQUEST_TYPE Ikev2ChildExchangeRequestTy
*IkePacket)
UINT32 Ikev2ChildExchangeRekeySpi
*IkePacket)
EFI_STATUS Ikev2ChildSaAssociateSpdEntr
IKEV2_CHILD_SA_SESSION
BOOLEAN Ikev2ValidateHeader (IN IKEV
*IkeSaSession, IN IKE_HEAD
IKEV2_SA_DATA * Ikev2InitializeSaData (IN
IKEV2_SESSION_COMMON
VOID Ikev2StoreSaData (IN
IKEV2_CHILD_SA_SESSION
VOID Ikev2ChildSaBeforeDecodePay
*SessionCommon, IN UINT8 *
UINTN PayloadSize, IN UINT8
VOID Ikev2ChildSaAfterEncodePaylo
*SessionCommon, IN UINT8 *
UINTN PayloadSize, IN UINT8
IKE_PAYLOAD * Ikev2GenerateSaPayload (IN IK
*SessionSaData, IN UINT8 Ne
IKE_SESSION_TYPE Type)
IKE_PAYLOAD * Ikev2GenerateIdPayload (IN
IKEV2_SESSION_COMMON
UINT8 NextPayload)
IKE_PAYLOAD * Ikev2GenerateCertIdPayload (I
IKEV2_SESSION_COMMON
UINT8 NextPayload, IN UINT8
CertSize)
IKE_PAYLOAD * Ikev2GenerateNoncePayload (I
IN UINTN NonceSize, IN UIN
IKE_PAYLOAD * Ikev2GenerateNotifyPayload (I
IN UINT8 NextPayload, IN UI
UINT16 MessageType, IN UIN
UINT8 *NotifyData, IN UINTN
IKE_PAYLOAD * Ikev2GenerateDeletePayload (I
IKEV2_SA_SESSION *IkeSaS
NextPayload, IN UINT8 SpiSiz
SpiNum, IN UINT8 *SpiBuf)
IKE_PAYLOAD * Ikev2GenerateCpPayload (IN I
*IkeSaSession, IN UINT8 Next
CfgType)
IKE_PAYLOAD * Ikev2PskGenerateAuthPayload
IKEV2_SA_SESSION *IkeSaS
IKE_PAYLOAD *IdPayload, I

NextPayload, IN BOOI
IKE_PAYLOAD * Ikev2CertGenerateAuth
IKEV2_SA_SESSION
IKE_PAYLOAD *IdPa
NextPayload, IN BOOI
*UefiPrivateKey, IN U
UINT8 *UefiKeyPwd,
IKE_PAYLOAD * Ikev2GenerateTsPayloa
IKEV2_CHILD_SA_S
NextPayload, IN BOOI
EFI_STATUS Ikev2ParserNotifyCook
IKE_PAYLOAD *IkeN
IKEV2_SA_SESSION
IKE_PAYLOAD * Ikev2GenerateCertifica
IKEV2_SA_SESSION
NextPayload, IN UINT
CertificateLen, IN UIN
BOOLEAN IsRequest)
EFI_STATUS Ikev2EncodePayload (I
IN OUT IKE_PAYLOA
EFI_STATUS Ikev2DecodePayload (I
IN OUT IKE_PAYLOA
EFI_STATUS Ikev2DecryptPacket (IN
IKEV2_SESSION_CO
OUT IKE_PACKET *I
IkeType)
EFI_STATUS Ikev2EncryptPacket (IN
IKEV2_SESSION_CO
OUT IKE_PACKET *I
EFI_STATUS Ikev2EncodePacket (IN
IKEV2_SESSION_CO
OUT IKE_PACKET *I
IkeType)
EFI_STATUS Ikev2DecodePacket (IN
IKEV2_SESSION_CO
OUT IKE_PACKET *I
IkeType)
VOID Ikev2OnPacketAccepte
IKEV2_SESSION_CO
IKE_PACKET *IkePac
EFI_STATUS Ikev2SendIkePacket (IN
*IkeUdpService, IN UI
IKE_PACKET *IkePac
VOID EFIAPI Ikev2LifetimeNotify (I
VOID *Context)
VOID EFIAPI Ikev2ResendNotify (IN
VOID *Context)
IKE_PAYLOAD * Ikev2GenerateKePaylo
IKEV2_SA_SESSION
NextPayload)
EFI_STATUS Ikev2MatchSpdEntry (I
EFI_IPSEC_CONFIG_
EFI_IPSEC_CONFIG_
VOID *Data, IN UINT

DataSize, IN VOID *Context)
BOOLEAN Ikev2IsSupportAlg (IN UINT16
UINT8 Type)
IKEV2_CHILD_SA_SESSION * Ikev2ChildSaSessionCreate (IN
IKEV2_SA_SESSION *IkeSaS
IKE_UDP_SERVICE *UdpSer
BOOLEAN Ikev2SaParseSaPayload (IN OU
IKEV2_SA_SESSION *IkeSaS
IKE_PAYLOAD *SaPayload, I
BOOLEAN Ikev2ChildSaParseSaPayload (I
IKEV2_CHILD_SA_SESSION
IKE_PAYLOAD *SaPayload, I
EFI_STATUS Ikev2SaGenerateKey (IN UINT
UINT8 *HashKey, IN UINTN I
OUT UINT8 *OutputKey, IN U
OutputKeyLength, IN PRF_DA
*Fragments, IN UINTN NumFr
VOID Ikev2ChildSaSessionSpdSelect
IKEV2_CHILD_SA_SESSION

Variables

IKE_ALG_GUID_INFO mIPsecEncrAlgInfo []

Detailed Description

The interfaces of IKE/Child session operations and payload related operations used by IKE Exchange Process.

Copyright (c) 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Ikev2IsSupportAlg().

#define IKEV2_SUPPORT_DH_ALGORITHM_NUM 2

Referenced by Ikev2IsSupportAlg().

#define IKEV2_SUPPORT_ENCRYPT_ALGORITHM_NUM 2

Referenced by Ikev2IsSupportAlg().

#define IKEV2_SUPPORT_PRF_ALGORITHM_NUM 1

Referenced by Ikev2IsSupportAlg().

Function Documentation

```
IKE_PAYLOAD* Ikev2CertGenerateAuthPayload ( IN IKEV2_SA_SESSION *   IkeSaSession,
                                            IN IKE_PAYLOAD *        IdPayload,
                                            IN UINT8                NextPayload,
                                            IN BOOLEAN              IsVerify,
                                            IN UINT8 *              UefiPrivateKey,
                                            IN UINTN                UefiPrivateKeyLen,
                                            IN UINT8 *              UefiKeyPwd,
                                            IN UINTN                UefiKeyPwdLen
                                          )
```

Generate a Authentication Payload for Certificate Auth method.

This function has two functions. One is creating a local Authentication Payload for sending and other is creating the remote Authentication data for verification when the IsVerify is TURE.

Parameters:

[in]	*IkeSaSession*	Pointer to IKEV2_SA_SESSION related to.
[in]	*IdPayload*	Pointer to the ID payload to be used for Authentication payload generation.
[in]	*NextPayload*	The type filled into the Authentication Payload next payload field.
[in]	*IsVerify*	If it is TURE, the Authentication payload is used for verification.
[in]	*UefiPrivateKey*	Pointer to the UEFI private key. Ignore it when verify the authenticate payload.
[in]	*UefiPrivateKeyLen*	The size of UefiPrivateKey in bytes. Ignore it when verify the authenticate payload.
[in]	*UefiKeyPwd*	Pointer to the password of UEFI private key. Ignore it when verify the authenticate payload.
[in]	*UefiKeyPwdLen*	The size of UefiKeyPwd in bytes.Ignore it when verify the authenticate payload.

Returns:

pointer to IKE Authentication payload for certification method.

Generate a Authentication Payload for Certificate Auth method.

This function has two functions. One is creating a local Authentication Payload for sending and other is creating the remote Authentication data for verification when the IsVerify is TURE.

Parameters:

[in]	*IkeSaSession*	Pointer to IKEV2_SA_SESSION related to.
[in]	*IdPayload*	Pointer to the ID payload to be used for Authentication payload generation.
[in]	*NextPayload*	The type filled into the Authentication Payload next payload field.
[in]	*IsVerify*	If it is TURE, the Authentication payload is used for verification.

[in]	*UefiPrivateKey*	Pointer to the UEFI private key. Ignore it when verify the authenticate payload.
[in]	*UefiPrivateKeyLen*	The size of UefiPrivateKey in bytes. Ignore it when verify the authenticate payload.
[in]	*UefiKeyPwd*	Pointer to the password of UEFI private key. Ignore it when verify the authenticate payload.
[in]	*UefiKeyPwdLen*	The size of UefiKeyPwd in bytes.Ignore it when verify the authenticate payload.

Returns:
pointer to IKE Authentication payload for Cerifitcation method.

References IKEV2_AUTH::AuthMethod, PRF_DATA_FRAGMENT::Data, PRF_DATA_FRAGMENT::DataSize, IKEV2_AUTH::Header, IkePayloadAlloc(), IkePayloadFree(), IKEV2_AUTH_METHOD_RSA, IKEV2_PAYLOAD_TYPE_AUTH, IpSecCryptoIoAuthDataWithCertificate(), IpSecCryptoIoHash(), IpSecCryptoIoHmac(), IpSecDumpBuf(), IpSecGetHmacDigestLength(), IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2AuthCertGenerator(), and Ikev2AuthCertParser().

UINT32 Ikev2ChildExchangeRekeySpi (IN IKE_PACKET * *IkePacket*)

This function finds the SPI from Create Child Sa Exchange Packet.

Parameters:
[in] *IkePacket* Pointer to IKE_PACKET to be searched.

Return values:
SPI number.

This function finds the SPI from Create Child SA Exchange Packet.

Parameters:
[in] *IkePacket* Pointer to IKE_PACKET to be searched.

Return values:
SPI number or 0 if it is not supported.

IKEV2_CREATE_CHILD_REQUEST_TYPE Ikev2ChildExchangeRequestType (IN IKE_PACKET * *IkePacket*)

This function is to parse a request IKE packet and return its request type. The request type is one of IKE CHILD SA creation, IKE SA rekeying and IKE CHILD SA rekeying.

Parameters:
[in] *IkePacket* IKE packet to be prased.

return the type of the IKE packet.

References IKE_PAYLOAD_BY_PACKET, IkeRequestTypeCreateChildSa, IkeRequestTypeRekeyChildSa, IkeRequestTypeRekeyIkeSa, IKEV2_NOTIFICATION_REKEY_SA, IKEV2_PAYLOAD_TYPE_NOTIFY, IKEV2_PAYLOAD_TYPE_TS_INIT, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2HandleChildSa().

```
VOID Ikev2ChildSaAfterEncodePayload ( IN UINT8 *   SessionCommon,
                                      IN UINT8 *   PayloadBuf,
                                      IN UINTN     PayloadSize,
                                      IN UINT8     PayloadType
                                    )
```

Routine Process after the encode payload.

Parameters:

[in]	*SessionCommon*	Pointer to ChildSa SessionCommon.
[in]	*PayloadBuf*	Pointer to the payload.
[in]	*PayloadSize*	Size of PayloadBuf in byte.
[in]	*PayloadType*	Type of Payload.

Routine Process after the payload encoding.

Parameters:

[in]	*SessionCommon*	Pointer to ChildSa SessionCommon.
[in]	*PayloadBuf*	Pointer to the payload.
[in]	*PayloadSize*	Size of PayloadBuf in byte.
[in]	*PayloadType*	Type of Payload.

Referenced by Ikev2ChildSaSessionAlloc().

```
EFI_STATUS                                  ( IN OUT IKEV2_CHILD_SA_SESSION *   ChildSaSession )
Ikev2ChildSaAssociateSpdEntry
```

Associate a SPD selector to the Child SA Session.

This function is called when the Child SA is not the first child SA of its IKE SA. It associate a SPD to this Child SA.

Parameters:

[in,out]	*ChildSaSession*	Pointer to the Child SA Session to be associated to a SPD selector.

Return values:

EFI_SUCCESS	Associate one SPD selector to this Child SA Session successfully.
EFI_NOT_FOUND	Can't find the related SPD selector.

References Ikev2MatchSpdEntry(), and IpSecVisitConfigData().

Referenced by Ikev2AuthCertParser(), and Ikev2AuthPskParser().

```
VOID Ikev2ChildSaBeforeDecodePayload ( IN UINT8 *   SessionCommon,
                                       IN UINT8 *   PayloadBuf,
                                       IN UINTN     PayloadSize,
                                       IN UINT8     PayloadType
                                     )
```

Routine process before the payload decoding.

Parameters:

[in]	*SessionCommon*	Pointer to ChildSa SessionCommon.
[in]	*PayloadBuf*	Pointer to the payload.
[in]	*PayloadSize*	Size of PayloadBuf in byte.
[in]	*PayloadType*	Type of Payload.

Referenced by Ikev2ChildSaSessionAlloc().

```
BOOLEAN Ikev2ChildSaParseSaPayload (  IN OUT IKEV2_CHILD_SA_SESSION *  ChildSaSession,
                                      IN IKE_PAYLOAD *                 SaPayload,
                                      IN UINT8                         Type
                                    )
```

Parse the received Authentication Exchange Packet.

This function parse the SA Payload and Key Payload to find out the cryptographic suite for the ESP and fill it into the Child SA Session's CommonSession->SaParams.

Parameters:

[in,out]	*ChildSaSession*	Pointer to IKEV2_CHILD_SA_SESSION related to this Authentication Exchange.
[in]	*SaPayload*	The received packet.
[in]	*Type*	The IKE header's flag of received packet .

Return values:

TRUE	If the SA proposal in Packet is acceptable.
FALSE	If the SA proposal in Packet is not acceptable.

References IKE_HEADER_FLAGS_INIT, IKE_HEADER_FLAGS_RESPOND, Ikev2ParseProposalData(), IPSEC_PROTO_IPSEC_ESP, IKEV2_PROPOSAL_DATA::NumTransforms, IKEV2_PROPOSAL_DATA::ProtocolId, and IKEV2_PROPOSAL_DATA::Spi.

Referenced by Ikev2AuthCertParser(), and Ikev2AuthPskParser().

```
IKEV2_CHILD_SA_SESSION* Ikev2ChildSaSessionAlloc ( IN IKE_UDP_SERVICE *    UdpService,
                                                   IN IKEV2_SA_SESSION *   IkeSaSession
                                                 )
```

Allocate Momery for IKEV2 Child SA Session.

Parameters:

[in]	*UdpService*	Pointer to IKE_UDP_SERVICE.
[in]	*IkeSaSession*	Pointer to IKEV2_SA_SESSION related to this Child SA Session.

Return values:

Pointer	of a new created IKEV2 Child SA Session.

Allocate memory for IKEV2 Child SA Session.

Parameters:

[in] *UdpService* Pointer to IKE_UDP_SERVICE.

[in] *IkeSaSession* Pointer to IKEV2_SA_SESSION related to this Child SA Session.

Return values:

Pointer of a new created IKEV2 Child SA Session or NULL.

References IKEV2_SESSION_COMMON::AfterEncodePayload, IKEV2_SESSION_COMMON::BeforeDecodePayload, IkeGenerateSpi(), IKEV2_CHILD_SA_SESSION::IkeSaSession, IKEV2_SESSION_COMMON::IkeSessionType, IkeSessionTypeChildSa, IKEV2_CHILD_SA_SESSION_SIGNATURE, Ikev2ChildSaAfterEncodePayload(), Ikev2ChildSaBeforeDecodePayload(), Ikev2ResendNotify(), IKEV2_SESSION_COMMON::IkeVer, IKEV2_SESSION_COMMON::LocalPeerIp, IKEV2_CHILD_SA_SESSION::LocalPeerSpi, IKEV2_CHILD_SA_SESSION::MessageId, IKEV2_SESSION_COMMON::Private, IKEV2_SESSION_COMMON::RemotePeerIp, IKEV2_SA_SESSION::SessionCommon, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_CHILD_SA_SESSION::Signature, IKEV2_SESSION_COMMON::TimeoutEvent, and IKEV2_SESSION_COMMON::UdpService.

Referenced by Ikev2ChildSaSessionCreate(), and Ikev2NegotiateChildSa().

IKEV2_CHILD_SA_SESSION*
Ikev2ChildSaSessionCreate (IN IKEV2_SA_SESSION * *IkeSaSession*,
IN IKE_UDP_SERVICE * *UdpService*
)

Generate a ChildSa Session and insert it into related IkeSaSession.

Parameters:

[in] *IkeSaSession* Pointer to related IKEV2_SA_SESSION.

[in] *UdpService* Pointer to related IKE_UDP_SERVICE.

Returns:

pointer of IKEV2_CHILD_SA_SESSION.

References IkeStateAuth, IkeStateCreateChild, IKEV2_DUMP_STATE, Ikev2ChildSaSessionAlloc(), Ikev2ChildSaSessionInsert(), Ikev2ChildSaSessionSpdSelectorCreate(), IKEV2_SESSION_COMMON::IsInitiator, IKEV2_CHILD_SA_SESSION::LocalPort, IKEV2_CHILD_SA_SESSION::NiBlkSize, IKEV2_CHILD_SA_SESSION::NiBlock, IKEV2_CHILD_SA_SESSION::NrBlkSize, IKEV2_CHILD_SA_SESSION::NrBlock, IKEV2_CHILD_SA_SESSION::ProtoId, IKEV2_CHILD_SA_SESSION::RemotePort, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_CHILD_SA_SESSION::Spd, and IKEV2_SESSION_COMMON::State.

Referenced by Ikev2HandleSa().

VOID Ikev2ChildSaSessionFree (IN IKEV2_CHILD_SA_SESSION * *ChildSaSession*)

Free the memory located for the specified IKEV2_CHILD_SA_SESSION.

Parameters:

[in] *ChildSaSession* Pointer to IKEV2_CHILD_SA_SESSION.

References Ikev2DhBufferFree(), and Ikev2SaSessionCommonFree().

Referenced by Ikev2ChildSaSessionReg(), Ikev2ChildSaSilentDelete(), Ikev2HandleSa(), Ikev2NegotiateChildSa(), Ikev2ResendNotify(), and Ikev2SaSessionFree().

```
VOID Ikev2ChildSaSessionInsert ( IN LIST_ENTRY *                SaSessionList,
                                 IN IKEV2_CHILD_SA_SESSION *    ChildSaSession
                               )
```

Insert a Child SA Session into the specified ChildSa list..

Parameters:

[in] *SaSessionList* Pointer to list to be inserted in.
[in] *ChildSaSession* Pointer to IKEV2_CHILD_SA_SESSION to be inserted.

Insert a Child SA Session into the specified ChildSa list.

Parameters:

[in] *SaSessionList* Pointer to list to be inserted in.
[in] *ChildSaSession* Pointer to IKEV2_CHILD_SA_SESSION to be inserted.

Referenced by Ikev2ChildSaSessionCreate(), Ikev2ChildSaSessionReg(), and Ikev2NegotiateChildSa().

```
IKEV2_CHILD_SA_SESSION*
Ikev2ChildSaSessionLookupByMid ( IN LIST_ENTRY *  SaSessionList,
                                 IN UINT32        Mid
                               )
```

Find the ChildSaSession by it's MessagId.

Parameters:

[in] *SaSessionList* Pointer to a ChildSaSession List.
[in] *Mid* The messageId used to search ChildSaSession.

Returns:

Pointer to IKEV2_CHILD_SA_SESSION.

Find the ChildSaSession by it's MessagId.

Parameters:

[in] *SaSessionList* Pointer to a ChildSaSession List.
[in] *Mid* The messageId used to search ChildSaSession.

Returns:

Pointer to IKEV2_CHILD_SA_SESSION or NULL.

References IKEV2_CHILD_SA_SESSION_BY_IKE_SA, and IKEV2_CHILD_SA_SESSION::MessageId.

```
IKEV2_CHILD_SA_SESSION*
Ikev2ChildSaSessionLookupBySpi ( IN LIST_ENTRY *  SaSessionList,
                                 IN UINT32        Spi
```

)

This function find the Child SA by the specified Spi.

This functin find a ChildSA session by searching the ChildSaSessionlist of the input IKEV2_SA_SESSION by specified MessageID.

Parameters:

[in]	*SaSessionList*	Pointer to List to be searched.
[in]	*Spi*	Specified SPI.

Returns:

Pointer to IKEV2_CHILD_SA_SESSION.

This function find the Child SA by the specified SPI.

This functin find a ChildSA session by searching the ChildSaSessionlist of the input IKEV2_SA_SESSION by specified MessageID.

Parameters:

[in]	*SaSessionList*	Pointer to List to be searched.
[in]	*Spi*	Specified SPI.

Returns:

Pointer to IKEV2_CHILD_SA_SESSION or NULL.

References IKEV2_CHILD_SA_SESSION_BY_IKE_SA, IKEV2_CHILD_SA_SESSION::LocalPeerSpi, and IKEV2_CHILD_SA_SESSION::RemotePeerSpi.

Referenced by Ikev2OnPacketSent().

EFI_STATUS Ikev2ChildSaSessionOnDeleting (IN IKEV2_CHILD_SA_SESSION * *ChildSaSession*)

Mark a specified Child SA Session as on deleting.

Parameters:

[in]	*ChildSaSession*	Pointer to IKEV2_CHILD_SA_SESSION.

Return values:

EFI_SUCCESS	Operation is successful.

VOID Ikev2ChildSaSessionReg (IN IKEV2_CHILD_SA_SESSION * *ChildSaSession*,
IN IPSEC_PRIVATE_DATA * *Private*
)

Register a established IKEv2 Child SA into IkeSaSession->ChildSaEstablishSessionList. If the there is IKEV2_CHILD_SA_SESSION with same remote peer IP, remove the old one then register the new one.

Parameters:

[in]	*ChildSaSession*	Pointer to IKEV2_CHILD_SA_SESSION to be registered.
[in]	*Private*	Pointer to IPSEC_PRAVATE_DATA.

References CHILD_SA_DEFAULT_LIFETIME, IKEV2_SA_SESSION::ChildSaEstablishSessionList, IKEV2_ESTABLISHED_CHILDSA_LIST, Ikev2ChildSaSessionFree(), Ikev2ChildSaSessionInsert(),

Ikev2ChildSaSessionRemove(), Ikev2LifetimeNotify(), Ikev2SessionCommonRefresh(), Ikev2StoreSaData(), and IKEV2_SESSION_COMMON::TimeoutEvent.

Referenced by Ikev2HandleSa().

```
IKEV2_CHILD_SA_SESSION* Ikev2ChildSaSessionRemove ( IN LIST_ENTRY *  SaSessionList,
                                                    IN UINT32        Spi,
                                                    IN UINT8         ListType
                                                  )
```

Remove the IKEV2_CHILD_SA_SESSION from IkeSaSessionList.

Parameters:

[in]	*SaSessionList*	The SA Session List to be iterated.
[in]	*Spi*	Spi used to identify the IKEV2_CHILD_SA_SESSION.
[in]	*ListType*	The type of the List to indicate whether it is a Established.

Returns:

The point to IKEV2_CHILD_SA_SESSION.

Remove the IKEV2_CHILD_SA_SESSION from IkeSaSessionList.

Parameters:

[in]	*SaSessionList*	The SA Session List to be iterated.
[in]	*Spi*	Spi used to identified the IKEV2_CHILD_SA_SESSION.
[in]	*ListType*	The type of the List to indicate whether it is a Established.

Returns:

The point to IKEV2_CHILD_SA_SESSION or NULL.

References IKEV2_CHILD_SA_SESSION_BY_DEL_SA, IKEV2_CHILD_SA_SESSION_BY_IKE_SA, IKEV2_DELET_CHILDSA_LIST, IKEV2_ESTABLISHED_CHILDSA_LIST, IKEV2_ESTABLISHING_CHILDSA_LIST, IKEV2_CHILD_SA_SESSION::LocalPeerSpi, and IKEV2_CHILD_SA_SESSION::RemotePeerSpi.

Referenced by Ikev2ChildSaSessionReg(), Ikev2ChildSaSilentDelete(), Ikev2HandleSa(), Ikev2OnPacketSent(), and Ikev2ResendNotify().

```
                                                IN OUT
VOID Ikev2ChildSaSessionSpdSelectorCreate (  IKEV2_CHILD_SA_SESSION  ChildSaSession  )
                                             *
```

Copy ChildSaSession->Spd->Selector to ChildSaSession->SpdSelector.

ChildSaSession->SpdSelector stores the real Spdselector for its SA. Sometime, the SpdSelector in ChildSaSession is more accurated or the scope is smaller than the one in ChildSaSession->Spd, especially for the tunnel mode.

Parameters:

[in,out]	*ChildSaSession*	Pointer to IKEV2_CHILD_SA_SESSION related to.

Referenced by Ikev2AuthCertParser(), Ikev2AuthPskParser(), and Ikev2ChildSaSessionCreate().

```
EFI_STATUS Ikev2ChildSaSilentDelete ( IN IKEV2_SA_SESSION *   IkeSaSession,
                                      IN UINT32               Spi
                                    )
```

Delete the specified established Child SA.

This function delete the Child SA directly and dont send the Information Packet to remote peer.

Parameters:

[in]	*IkeSaSession*	Pointer to a IKE SA Session used to be searched for.
[in]	*Spi*	SPI used to find the Child SA.

Return values:

EFI_NOT_FOUND	Pointer of IKE SA Session is NULL.
EFI_NOT_FOUND	There is no specified Child SA related with the input SPI under this IKE SA Session.
EFI_SUCCESS	Delete the Child SA successfully.

Delete the specified established Child SA.

This function delete the Child SA directly and don't send the Information Packet to remote peer.

Parameters:

[in]	*IkeSaSession*	Pointer to a IKE SA Session used to be searched for.
[in]	*Spi*	SPI used to find the Child SA.

Return values:

EFI_NOT_FOUND	Pointer of IKE SA Session is NULL.
EFI_NOT_FOUND	There is no specified Child SA related with the input SPI under this IKE SA Session.
EFI_SUCCESS	Delete the Child SA successfully.

References EfiIpSecConfigGetNextSelector(), EfiIpSecConfigSetData(), IKEV2_ESTABLISHED_CHILDSA_LIST, Ikev2ChildSaSessionFree(), Ikev2ChildSaSessionRemove(), _IPSEC_PRIVATE_DATA::IpSecConfig, IKEV2_CHILD_SA_SESSION::LocalPeerSpi, and IKEV2_CHILD_SA_SESSION::RemotePeerSpi.

Referenced by Ikev2InfoParser(), Ikev2OnPacketSent(), Ikev2ResendNotify(), and Ikev2SaSessionFree().

```
EFI_STATUS Ikev2DecodePacket ( IN IKEV2_SESSION_COMMON *   SessionCommon,
                               IN OUT IKE_PACKET *         IkePacket,
                               IN UINTN                    IkeType
                             )
```

Decode the IKE packet.

This function first decrypts the IKE packet if needed , then separats the whole IKE packet from the IkePacket->PayloadBuf into IkePacket payload list.

Parameters:

[in]	*SessionCommon*	Pointer to IKEV1_SESSION_COMMON containing some parameter used by IKE packet decoding.
[in,out]	*IkePacket*	

		The IKE Packet to be decoded on input, and the decoded result on return.
[in]	*IkeType*	The type of IKE. IKE_SA_TYPE, IKE_INFO_TYPE and IKE_CHILD_TYPE are supportted.

Return values:

EFI_SUCCESS	The IKE packet is decoded successfull.
Otherwise	The IKE packet decoding is failed.

Decode the IKE packet.

This function first decrypts the IKE packet if needed , then separates the whole IKE packet from the IkePacket->PayloadBuf into IkePacket payload list.

Parameters:

[in]	*SessionCommon*	Pointer to IKEV1_SESSION_COMMON containing some parameter used by IKE packet decoding.
[in,out]	*IkePacket*	The IKE Packet to be decoded on input, and the decoded result on return.
[in]	*IkeType*	The type of IKE. IKE_SA_TYPE, IKE_INFO_TYPE and IKE_CHILD_TYPE are supported.

Return values:

EFI_SUCCESS	The IKE packet is decoded successfully.
Otherwise	The IKE packet decoding is failed.

References ClearAllPayloads(), IKE_PACKET_APPEND_PAYLOAD, IkeHdrNetToHost(), IkePayloadAlloc(), IkeStateAuth, IKEV2_EXCHANGE_TYPE_INFO, IKEV2_EXCHANGE_TYPE_INIT, IKEV2_PAYLOAD_TYPE_NONE, IKEV2_SA_SESSION_FROM_COMMON, Ikev2DecodePayload(), Ikev2DecryptPacket(), IKEV2_SA_SESSION::InitPacket, IKEV2_SA_SESSION::InitPacketSize, IPSEC_DUMP_BUF, IKE_PAYLOAD::IsPayloadBufExt, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, IKEV2_SA_SESSION::RespPacket, and IKEV2_SA_SESSION::RespPacketSize.

Referenced by Ikev2HandleChildSa(), Ikev2HandleInfo(), and Ikev2HandleSa().

```
EFI_STATUS Ikev2DecodePayload ( IN UINT8 *              SessionCommon,
                                IN OUT IKE_PAYLOAD *    IkePayload
                              )
```

The general interface of decode Payload.

This function convert the received Payload into internal structure.

Parameters:

[in]	*SessionCommon*	Pointer to IKE Session Common to use for decoding.
[in,out]	*IkePayload*	Pointer to IKE payload to be decode as input, and store the decoded result as output.

Return values:

EFI_INVALID_PARAMETER	Meet error when decode the SA payload.

EFI_SUCCESS	Decode successfully.

The general interface for decoding Payload.

This function converts the received Payload into internal structure.

Parameters:

`[in]`	*SessionCommon*	Pointer to IKE Session Common used for decoding.
`[in,out]`	*IkePayload*	Pointer to IKE payload to be decoded as input, and store the decoded result as output.

Return values:

EFI_INVALID_PARAMETER	Meet error when decoding the SA payload.
EFI_SUCCESS	Decoded successfully.

References IKEV2_CFG_ATTRIBUTES::AttritType, IKEV2_KEY_EXCHANGE::DhGroup, TRAFFIC_SELECTOR::EndPort, IKEV2_DUMP_PAYLOAD, IKEV2_PAYLOAD_TYPE_AUTH, IKEV2_PAYLOAD_TYPE_CP, IKEV2_PAYLOAD_TYPE_DELETE, IKEV2_PAYLOAD_TYPE_ID_INIT, IKEV2_PAYLOAD_TYPE_ID_RSP, IKEV2_PAYLOAD_TYPE_KE, IKEV2_PAYLOAD_TYPE_NOTIFY, IKEV2_PAYLOAD_TYPE_SA, IKEV2_PAYLOAD_TYPE_TS_INIT, IKEV2_PAYLOAD_TYPE_TS_RSP, Ikev2DecodeSa(), IKEV2_NOTIFY::MessageType, IKEV2_DELETE::NumSpis, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, TRAFFIC_SELECTOR::SelecorLen, TRAFFIC_SELECTOR::StartPort, IKEV2_TS::TSNumbers, and IKEV2_CFG_ATTRIBUTES::ValueLength.

Referenced by Ikev2DecodePacket().

```
EFI_STATUS Ikev2DecryptPacket ( IN IKEV2_SESSION_COMMON *   SessionCommon,
                                IN OUT IKE_PACKET *         IkePacket,
                                IN OUT UINTN                IkeType
                              )
```

Decrypt IKE packet.

This function decrpt the Encrypted IKE packet and put the result into IkePacket->PayloadBuf.

Parameters:

`[in]`	*SessionCommon*	Pointer to IKEV2_SESSION_COMMON containing some parameter used during decrypting.
`[in,out]`	*IkePacket*	Point to IKE_PACKET to be decrypted as input, and the decrypted reslult as output.
`[in,out]`	*IkeType*	The type of IKE. IKE_SA_TYPE, IKE_INFO_TYPE and IKE_CHILD_TYPE are supportted.

Return values:

EFI_INVALID_PARAMETER	If the IKE packet length is zero or the IKE packet length is not Algorithm Block Size alignment.
EFI_SUCCESS	Decrypt IKE packet successfully.

Decrypt IKE packet.

This function decrypts the Encrypted IKE packet and put the result into IkePacket->PayloadBuf.

Parameters:

[in]	*SessionCommon*	Pointer to IKEV2_SESSION_COMMON containing some parameter used during decrypting.
[in,out]	*IkePacket*	Pointer to IKE_PACKET to be decrypted as input, and the decrypted result as output.
[in,out]	*IkeType*	The type of IKE. IKE_SA_TYPE, IKE_INFO_TYPE and IKE_CHILD_TYPE are supportted.

Return values:

EFI_INVALID_PARAMETER	If the IKE packet length is zero or the IKE packet length is not aligned with Algorithm Block Size
EFI_SUCCESS	Decrypt IKE packet successfully.

References HASH_DATA_FRAGMENT::Data, HASH_DATA_FRAGMENT::DataSize, IKEV2_SA_PARAMS::EncAlgId, IkeHdrHostToNet(), IKEV2_SA_SESSION::IkeKeys, IKEV2_CHILD_SA_SESSION::IkeSaSession, IkeSessionTypeChildSa, IkeSessionTypeIkeSa, IKEV2_CHILD_SA_SESSION_FROM_COMMON, IKEV2_PAYLOAD_TYPE_ENCRYPT, IKEV2_SA_SESSION_FROM_COMMON, IKEV2_SA_PARAMS::IntegAlgId, IPSEC_DUMP_BUF, IpSecCryptoIoDecrypt(), IpSecCryptoIoHmac(), IpSecGetEncryptBlockSize(), IpSecGetIcvLength(), IKEV2_SESSION_COMMON::SaParams, IKEV2_SA_SESSION::SessionCommon, IKEV2_SESSION_KEYS::SkAiKey, IKEV2_SESSION_KEYS::SkAiKeySize, IKEV2_SESSION_KEYS::SkArKey, IKEV2_SESSION_KEYS::SkArKeySize, IKEV2_SESSION_KEYS::SkEiKey, IKEV2_SESSION_KEYS::SkEiKeySize, IKEV2_SESSION_KEYS::SkErKey, and IKEV2_SESSION_KEYS::SkErKeySize.

Referenced by Ikev2DecodePacket().

```
VOID Ikev2DhBufferFree ( IN IKEV2_DH_BUFFER *   DhBuffer   )
```

Free the specified DhBuffer.

Parameters:

[in]	*DhBuffer*	Pointer to IKEV2_DH_BUFFER to be freed.

Referenced by Ikev2ChildSaSessionFree(), and Ikev2SaSessionFree().

```
EFI_STATUS Ikev2EncodePacket ( IN IKEV2_SESSION_COMMON *   SessionCommon,
                               IN OUT IKE_PACKET *         IkePacket,
                               IN UINTN                    IkeType
                             )
```

Encode the IKE packet.

This function put all Payloads into one payload then encrypt it if needed.

Parameters:

[in]	*SessionCommon*	Pointer to IKEV2_SESSION_COMMON containing some parameter used during IKE packet encoding.
[in,out]	*IkePacket*	Pointer to IKE_PACKET to be encoded as input, and the encoded reslult as output.
[in]	*IkeType*	The type of IKE. IKE_SA_TYPE, IKE_INFO_TYPE and IKE_CHILD_TYPE are supportted.

Return values:

EFI_SUCCESS	Encode IKE packet successfully.
Otherwise	Encode IKE packet failed.

Encode the IKE packet.

This function puts all Payloads into one payload then encrypt it if needed.

Parameters:

`[in]`	*SessionCommon*	Pointer to IKEV2_SESSION_COMMON containing some parameter used during IKE packet encoding.
`[in,out]`	*IkePacket*	Pointer to IKE_PACKET to be encoded as input, and the encoded result as output.
`[in]`	*IkeType*	The type of IKE. IKE_SA_TYPE, IKE_INFO_TYPE and IKE_CHILD_TYPE are supportted.

Return values:

EFI_SUCCESS	Encode IKE packet successfully.
Otherwise	Encode IKE packet failed.

References IKE_PAYLOAD_BY_PACKET, IkeHdrHostToNet(), IkeStateAuth, IKEV2_EXCHANGE_TYPE_INIT, IKEV2_SA_SESSION_FROM_COMMON, Ikev2EncodePayload(), Ikev2EncryptPacket(), IKEV2_SA_SESSION::InitPacket, IKEV2_SA_SESSION::InitPacketSize, IKE_PAYLOAD::PayloadBuf, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, IKEV2_SA_SESSION::RespPacket, and IKEV2_SA_SESSION::RespPacketSize.

Referenced by IkeNetbufFromPacket().

```
EFI_STATUS Ikev2EncodePayload ( IN UINT8 *              SessionCommon,
                                IN OUT IKE_PAYLOAD *    IkePayload
                              )
```

General interface of payload encoding.

This function encode the internal data structure into payload which is defined in RFC 4306. The IkePayload->PayloadBuf used to store both the input payload and converted payload. Only the SA payload use the interal structure to store the attribute. Other payload use structure which is same with the RFC defined, for this kind payloads just do host order to network order change of some fields.

Parameters:

`[in]`	*SessionCommon*	Pointer to IKE Session Common used to encode the payload.
`[in,out]`	*IkePayload*	Pointer to IKE payload to be encode as input, and store the encoded result as output.

Return values:

EFI_INVALID_PARAMETER	Meet error when encode the SA payload.
EFI_SUCCESS	Encode successfully.

General interface of payload encoding.

This function encodes the internal data structure into payload which is defined in RFC 4306. The IkePayload->PayloadBuf is used to store both the input payload and converted payload. Only the SA payload use the interal structure to store the attribute. Other payload use structure which is same with the RFC defined,

for this kind payloads just do host order to network order change of some fields.

Parameters:

[in]	*SessionCommon*	Pointer to IKE Session Common used to encode the payload.
[in,out]	*IkePayload*	Pointer to IKE payload to be encoded as input, and store the encoded result as output.

Return values:

EFI_INVALID_PARAMETER	Meet error when encoding the SA payload.
EFI_SUCCESS	Encoded successfully.

References IKEV2_CFG_ATTRIBUTES::AttritType, IKEV2_KEY_EXCHANGE::DhGroup, TRAFFIC_SELECTOR::EndPort, IKEV2_DUMP_PAYLOAD, IKEV2_PAYLOAD_TYPE_AUTH, IKEV2_PAYLOAD_TYPE_CP, IKEV2_PAYLOAD_TYPE_DELETE, IKEV2_PAYLOAD_TYPE_ID_INIT, IKEV2_PAYLOAD_TYPE_ID_RSP, IKEV2_PAYLOAD_TYPE_KE, IKEV2_PAYLOAD_TYPE_NOTIFY, IKEV2_PAYLOAD_TYPE_SA, IKEV2_PAYLOAD_TYPE_TS_INIT, IKEV2_PAYLOAD_TYPE_TS_RSP, Ikev2EncodeSa(), IKEV2_NOTIFY::MessageType, IKEV2_DELETE::NumSpis, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, TRAFFIC_SELECTOR::SelecorLen, TRAFFIC_SELECTOR::StartPort, IKEV2_TS::TSNumbers, and IKEV2_CFG_ATTRIBUTES::ValueLength.

Referenced by Ikev2EncodePacket().

```
EFI_STATUS Ikev2EncryptPacket ( IN IKEV2_SESSION_COMMON *   SessionCommon,
                                IN OUT IKE_PACKET *         IkePacket
                              )
```

Encrypt IKE packet.

This function encrypt IKE packet before sending it. The Encrypted IKE packet is put in to IKEV2 Encrypted Payload.

Parameters:

[in]	*SessionCommon*	Pointer to IKEV2_SESSION_COMMON related to the IKE packet.
[in,out]	*IkePacket*	Pointer to IKE packet to be encrypted.

Return values:

EFI_SUCCESS	Operation is successful.
Others	OPeration is failed.

Encrypt IKE packet.

This function encrypt IKE packet before sending it. The Encrypted IKE packet is put in to IKEV2 Encrypted Payload.

Parameters:

[in]	*SessionCommon*	Pointer to IKEV2_SESSION_COMMON related to the IKE packet.
[in,out]	*IkePacket*	Pointer to IKE packet to be encrypted.

Return values:

EFI_SUCCESS	Operation is successful.
Others	Operation is failed.

References ClearAllPayloads(), HASH_DATA_FRAGMENT::Data, HASH_DATA_FRAGMENT::DataSize, IKEV2_SA_PARAMS::EncAlgId, IKE_PACKET_APPEND_PAYLOAD, IKE_PAYLOAD_BY_PACKET, IkeGenerateIv(), IkeHdrHostToNet(), IKEV2_SA_SESSION::IkeKeys, IkePayloadAlloc(), IKEV2_CHILD_SA_SESSION::IkeSaSession, IkeSessionTypeChildSa, IkeSessionTypeIkeSa, IKEV2_CHILD_SA_SESSION_FROM_COMMON, IKEV2_PAYLOAD_TYPE_ENCRYPT, IKEV2_SA_SESSION_FROM_COMMON, IKEV2_SA_PARAMS::IntegAlgId, IPSEC_DUMP_BUF, IpSecCryptoIoEncrypt(), IpSecCryptoIoHmac(), IpSecGetEncryptBlockSize(), IpSecGetIcvLength(), IKE_PAYLOAD::PayloadBuf, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, IKEV2_SESSION_COMMON::SaParams, IKEV2_SA_SESSION::SessionCommon, IKEV2_SESSION_KEYS::SkAiKey, IKEV2_SESSION_KEYS::SkAiKeySize, IKEV2_SESSION_KEYS::SkArKey, IKEV2_SESSION_KEYS::SkArKeySize, IKEV2_SESSION_KEYS::SkEiKey, IKEV2_SESSION_KEYS::SkEiKeySize, IKEV2_SESSION_KEYS::SkErKey, and IKEV2_SESSION_KEYS::SkErKeySize.

Referenced by Ikev2EncodePacket().

```
IKE_PAYLOAD* Ikev2GenerateCertIdPayload ( IN IKEV2_SESSION_COMMON *   CommonSession,
                                          IN UINT8                    NextPayload,
                                          IN UINT8 *                  InCert,
                                          IN UINTN                    CertSize
                                        )
```

Generate a ID payload.

Parameters:

[in]	*CommonSession*	Pointer to IKEV2_SESSION_COMMON related to ID payload.
[in]	*NextPayload*	The payload type presented in the NextPayload field of ID Payload header.
[in]	*InCert*	Pointer to the Certificate which distinguished name will be added into the Id payload.
[in]	*CertSize*	Size of the Certificate.

Return values:

Pointer to ID IKE payload.

Generate a ID payload.

Parameters:

[in]	*CommonSession*	Pointer to IKEV2_SESSION_COMMON related to ID payload.
[in]	*NextPayload*	The payload type presented in the NextPayload field of ID Payload header.
[in]	*InCert*	Pointer to the Certificate which distinguished name will be added into the Id payload.
[in]	*CertSize*	Size of the Certificate.

Return values:

Pointer to ID IKE payload.

References IKEV2_ID::Header, IKEV2_ID::IdType, IkePayloadAlloc(), IKEV2_PAYLOAD_TYPE_ID_INIT, IKEV2_PAYLOAD_TYPE_ID_RSP, IpSecCryptoIoGetSubjectFromCert(), IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2AuthCertGenerator().

```
IKE_PAYLOAD* Ikev2GenerateCertificatePayload (
    IN IKEV2_SA_SESSION *   IkeSaSession,
    IN UINT8                NextPayload,
    IN UINT8 *              Certificate,
    IN UINTN                CertificateLen,
    IN UINT8                EncodeType,
    IN BOOLEAN              IsRequest
)
```

Generate the Certificate payload or Certificate Request Payload.

Since the Certificate Payload structure is same with Certificate Request Payload, the only difference is that one contains the Certificate Data, other contains the acceptable certificateion CA. This function generate Certificate payload or Certificate Request Payload defined in RFC 4306, but all the fields in the payload are still in host order and need call Ikev2EncodePayload() to convert those fields from the host order to network order beforing sending it.

Parameters:

[in]	*IkeSaSession*	Pointer to IKE SA Session to be used of Delete payload generation.
[in]	*NextPayload*	The next paylaod type in NextPayload field of the Delete payload.
[in]	*Certificate*	Pointer of buffer contains the certification data.
[in]	*CertificateLen*	The length of Certificate in byte.
[in]	*EncodeType*	Specified the Certificate Encodeing which is defined in RFC 4306.
[in]	*IsRequest*	To indicate create Certificate Payload or Certificate Request Payload. If it is TURE, create Certificate Request Payload. Otherwise, create Certificate Payload.

Return values:

a Pointer to IKE Payload whose payload buffer containing the Certificate payload or Certificated Request payload.

Generate the Certificate payload or Certificate Request Payload.

Since the Certificate Payload structure is same with Certificate Request Payload, the only difference is that one contains the Certificate Data, other contains the acceptable certificateion CA. This function generate Certificate payload or Certificate Request Payload defined in RFC 4306, but all the fields in the payload are still in host order and need call Ikev2EncodePayload() to convert those fields from the host order to network order beforing sending it.

Parameters:

[in]	*IkeSaSession*	Pointer to IKE SA Session to be used of Delete payload generation.
[in]	*NextPayload*	The next paylaod type in NextPayload field of the Delete payload.
[in]	*Certificate*	Pointer of buffer contains the certification data.
[in]	*CertificateLen*	The length of Certificate in byte.
[in]	*EncodeType*	Specified the Certificate Encodeing which is defined in RFC 4306.
[in]	*IsRequest*	To indicate create Certificate Payload or Certificate Request Payload. If it is TURE, create Certificate Payload. Otherwise, create Certificate Request Payload.

Return values:

a Pointer to IKE Payload whose payload buffer containing the Certificate payload or Certificated Request payload.

References IKEV2_CERT::CertEncoding, HASH_DATA_FRAGMENT::Data, HASH_DATA_FRAGMENT::DataSize, IKEV2_CERT::Header, IKE_AALG_SHA1HMAC, IkePayloadAlloc(), IKEV2_PAYLOAD_TYPE_CERT, IKEV2_PAYLOAD_TYPE_CERTREQ, IpSecCryptoIoGetPublicKeyFromCert(), IpSecCryptoIoHash(), IpSecGetHmacDigestLength(), IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2AuthCertGenerator(), and Ikev2InitCertGenerator().

```
IKE_PAYLOAD* Ikev2GenerateCpPayload ( IN IKEV2_SA_SESSION * IkeSaSession,
                                      IN UINT8 NextPayload,
                                      IN UINT8 CfgType
                                    )
```

Generate the Configuration payload.

This function generates a configuration payload defined in RFC 4306, but all the fields in this payload are still in host order and need call Ikev2EncodePayload() to convert those fields from the host order to network order beforing sending it.

Parameters:

[in]	*IkeSaSession*	Pointer to IKE SA Session to be used for Delete payload generation.
[in]	*NextPayload*	The next paylaod type in NextPayload field of the Delete payload.
[in]	*CfgType*	The attribute type in the Configuration attribute.

Return values:

Pointer to IKE CP Payload.

Generate the Configuration payload.

This function generate configuration payload defined in RFC 4306, but all the fields in this payload are still in host order and need call Ikev2EncodePayload() to convert those fields from the host order to network order beforing sending it.

Parameters:

[in]	*IkeSaSession*	Pointer to IKE SA Session to be used for Delete payload generation.
[in]	*NextPayload*	The next paylaod type in NextPayload field of the Delete payload.
[in]	*CfgType*	The attribute type in the Configuration attribute.

Return values:

Pointer to IKE CP Payload.

References IKEV2_CFG_ATTRIBUTES::AttritType, IKEV2_CFG::CfgType, IKEV2_CFG::Header, IkePayloadAlloc(), IKEV2_CFG_TYPE_REQUEST, IKEV2_PAYLOAD_TYPE_CP, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, and IKEV2_CFG_ATTRIBUTES::ValueLength.

Referenced by Ikev2AuthCertGenerator(), and Ikev2AuthPskGenerator().

```
IKE_PAYLOAD* Ikev2GenerateDeletePayload (
    IN IKEV2_SA_SESSION *   IkeSaSession,
    IN UINT8                NextPayload,
    IN UINT8                SpiSize,
    IN UINT16               SpiNum,
    IN UINT8 *              SpiBuf
)
```

Generate the Delete payload.

Since the structure of Delete payload which defined in RFC 4306 is simple, there is no internal data structure for Delete payload. This function generate Delete payload defined in RFC 4306, but all the fields in this payload are still in host order and need call Ikev2EncodePayload() to convert those fields from the host order to network order beforing sending it.

Parameters:

[in]	*IkeSaSession*	Pointer to IKE SA Session to be used of Delete payload generation.
[in]	*NextPayload*	The next paylaod type in NextPayload field of the Delete payload.
[in]	*SpiSize*	Size of the SPI in SPI size field of the Delete Payload.
[in]	*SpiNum*	Number of SPI in NumofSPIs field of the Delete Payload.
[in]	*SpiBuf*	Pointer to buffer contains the SPI value.

Return values:

Pointer to IKE Delete Payload.

Generate the Delete payload.

Since the structure of Delete payload which defined in RFC 4306 is simple, there is no internal data structure for Delete payload. This function generate Delete payload defined in RFC 4306, but all the fields in this payload are still in host order and need call Ikev2EncodePayload() to convert those fields from the host order to network order beforing sending it.

Parameters:

[in]	*IkeSaSession*	Pointer to IKE SA Session to be used of Delete payload generation.
[in]	*NextPayload*	The next paylaod type in NextPayload field of the Delete payload.
[in]	*SpiSize*	Size of the SPI in SPI size field of the Delete Payload.
[in]	*SpiNum*	Number of SPI in NumofSPIs field of the Delete Payload.
[in]	*SpiBuf*	Pointer to buffer contains the SPI value.

Return values:

a Pointer of IKE Delete Payload.

References IKEV2_DELETE::Header, IkePayloadAlloc(), IKEV2_PAYLOAD_TYPE_DELETE, IPSEC_PROTO_IPSEC_ESP, IPSEC_PROTO_ISAKMP, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKEV2_DELETE::NumSpis, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, IKEV2_DELETE::ProtocolId, and IKEV2_DELETE::SpiSize.

Referenced by Ikev2InfoGenerator().

```
IKE_PAYLOAD* Ikev2GenerateIdPayload ( IN IKEV2_SESSION_COMMON *   CommonSession,
                                      IN UINT8                    NextPayload
                                    )
```

Generate a ID payload.

Parameters:

[in]	*CommonSession*	Pointer to IKEV2_SESSION_COMMON related to ID payload.
[in]	*NextPayload*	The payload type presented in the NextPayload field of ID Payload header.

Return values:

Pointer to ID IKE payload.

Generate a ID payload.

Parameters:

[in]	*CommonSession*	Pointer to IKEV2_SESSION_COMMON related to ID payload.
[in]	*NextPayload*	The payload type presented in the NextPayload field of ID Payload header.

Return values:

Pointer to ID IKE payload.

References IKEV2_ID::Header, IKEV2_ID::IdType, IkePayloadAlloc(), IKEV2_ID_TYPE_IPV4_ADDR, IKEV2_ID_TYPE_IPV6_ADDR, IKEV2_PAYLOAD_TYPE_ID_INIT, IKEV2_PAYLOAD_TYPE_ID_RSP, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2AuthPskGenerator().

```
IKE_PAYLOAD* Ikev2GenerateKePayload ( IN OUT IKEV2_SA_SESSION *   IkeSaSession,
                                      IN UINT8                    NextPayload
                                    )
```

Generate a Key Exchange payload according to the DH group type and save the public Key into IkeSaSession IkeKey field.

Parameters:

[in,out]	*IkeSaSession*	Pointer of the IKE_SA_SESSION.
[in]	*NextPayload*	The payload type presented in the NextPayload field of Key Exchange Payload header.

Return values:

Pointer to Key IKE payload.

Generate a Key Exchange payload according to the DH group type and save the public Key into IkeSaSession IkeKey field.

Parameters:

[in,out]	*IkeSaSession*	Pointer of the IKE_SA_SESSION.
[in]	*NextPayload*	The payload type presented in the NextPayload field of Key Exchange Payload header.

Return values:

Pointer to Key IKE payload.

References IKEV2_SESSION_KEYS::DhBuffer, IKEV2_KEY_EXCHANGE::DhGroup, IKEV2_DH_BUFFER::GxBuffer, IKEV2_DH_BUFFER::GxSize, IKEV2_KEY_EXCHANGE::Header, IkePayloadAlloc(), IKEV2_PAYLOAD_TYPE_KE, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2InitPskGenerator().

```
IKE_PAYLOAD* Ikev2GenerateNoncePayload ( IN UINT8 *   NonceBuf,
                                         IN UINTN     NonceSize,
                                         IN UINT8     NextPayload
                                       )
```

Generate a Nonce payload contenting the input parameter NonceBuf.

Parameters:

- [in] *NonceBuf* The nonce buffer content the whole Nonce payload block except the payload header.
- [in] *NonceSize* The buffer size of the NonceBuf
- [in] *NextPayload* The payload type presented in the NextPayload field of Nonce Payload header.

Return values:

Pointer to Nonce IKE paload.

Generate a Nonce payload containing the input parameter NonceBuf.

Parameters:

- [in] *NonceBuf* The nonce buffer contains the whole Nonce payload block except the payload header.
- [in] *NonceSize* The buffer size of the NonceBuf
- [in] *NextPayload* The payload type presented in the NextPayload field of Nonce Payload header.

Return values:

Pointer to Nonce IKE paload.

References IKEV2_NONCE::Header, IkePayloadAlloc(), IKEV2_PAYLOAD_TYPE_NONCE, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2InitPskGenerator().

```
IKE_PAYLOAD* Ikev2GenerateNotifyPayload ( IN UINT8     ProtocolId,
                                          IN UINT8     NextPayload,
                                          IN UINT8     SpiSize,
                                          IN UINT16    MessageType,
                                          IN UINT8 *   SpiBuf,
                                          IN UINT8 *   NotifyData,
```

IN UINTN *NotifyDataSize*
)

Generate the Notify payload.

Since the structure of Notify payload which defined in RFC 4306 is simple, so there is no internal data structure for Notify payload. This function generate Notify payload defined in RFC 4306, but all the fields in this payload are still in host order and need call Ikev2EncodePayload() to convert those fields from the host order to network order beforing sending it.

Parameters:

[in]	*ProtocolId*	The protocol type ID. For IKE_SA it MUST be one (1). For IPsec SAs it MUST be neither (2) for AH or (3) for ESP.
[in]	*NextPayload*	The next paylaod type in NextPayload field of the Notify payload.
[in]	*SpiSize*	Size of the SPI in SPI size field of the Notify Payload.
[in]	*MessageType*	The message type in NotifyMessageType field of the Notify Payload.
[in]	*SpiBuf*	Pointer to buffer contains the SPI value.
[in]	*NotifyData*	Pointer to buffer contains the notification data.
[in]	*NotifyDataSize*	The size of NotifyData in bytes.

Return values:

Pointer to IKE Notify Payload.

Generate the Notify payload.

Since the structure of Notify payload which defined in RFC 4306 is simple, so there is no internal data structure for Notify payload. This function generate Notify payload defined in RFC 4306, but all the fields in this payload are still in host order and need call Ikev2EncodePayload() to convert those fields from the host order to network order beforing sending it.

Parameters:

[in]	*ProtocolId*	The protocol type ID. For IKE_SA it MUST be one (1). For IPsec SAs it MUST be neither (2) for AH or (3) for ESP.
[in]	*NextPayload*	The next paylaod type in NextPayload field of the Notify payload.
[in]	*SpiSize*	Size of the SPI in SPI size field of the Notify Payload.
[in]	*MessageType*	The message type in NotifyMessageType field of the Notify Payload.
[in]	*SpiBuf*	Pointer to buffer contains the SPI value.
[in]	*NotifyData*	Pointer to buffer contains the notification data.
[in]	*NotifyDataSize*	The size of NotifyData in bytes.

Return values:

Pointer to IKE Notify Payload.

References IKEV2_NOTIFY::Header, IkePayloadAlloc(), IKEV2_PAYLOAD_TYPE_NOTIFY, IKEV2_NOTIFY::MessageType, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, IKEV2_NOTIFY::ProtocolId, and IKEV2_NOTIFY::SpiSize.

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), Ikev2CreateChildGenerator(), and Ikev2InitPskGenerator().

IKE_PAYLOAD* Ikev2GenerateSaPayload (IN IKEV2_SA_DATA * *SessionSaData*,

```
                                IN UINT8                  NextPayload,
                                IN IKE_SESSION_TYPE       Type
                              )
```

Generate Ikev2 SA payload according to SessionSaData

Parameters:

[in]	*SessionSaData*	The data used in SA payload.
[in]	*NextPayload*	The payload type presented in NextPayload field of SA Payload header.
[in]	*Type*	The SA type. It MUST be neither (1) for IKE_SA or (2) for CHILD_SA or (3) for INFO.

Return values:

a Pointer to SA IKE payload.

Generate Ikev2 SA payload according to SessionSaData

Parameters:

[in]	*SessionSaData*	The data used in SA payload.
[in]	*NextPayload*	The payload type presented in NextPayload field of SA Payload header.
[in]	*Type*	The SA type. It MUST be neither (1) for IKE_SA or (2) for CHILD_SA or (3) for INFO.

Return values:

a Pointer to SA IKE payload.

References IKEV2_SA::Header, IkePayloadAlloc(), IkeSessionTypeIkeSa, IKEV2_PAYLOAD_TYPE_SA, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKE_PAYLOAD::PayloadType, and IKEV2_SA_DATA::SaHeader.

Referenced by Ikev2AuthCertGenerator(), Ikev2AuthPskGenerator(), and Ikev2InitPskGenerator().

```
IKE_PAYLOAD* Ikev2GenerateTsPayload ( IN IKEV2_CHILD_SA_SESSION *  ChildSa,
                                      IN UINT8                     NextPayload,
                                      IN BOOLEAN                   IsTunnel
                                    )
```

Generate TS payload.

This function generates TSi or TSr payload according to type of next payload. If the next payload is Responder TS, gereate TSi Payload. Otherwise, generate TSr payload

Parameters:

[in]	*ChildSa*	Pointer to IKEV2_CHILD_SA_SESSION related to this TS payload.
[in]	*NextPayload*	The payload type presented in the NextPayload field of ID Payload header.
[in]	*IsTunnel*	It indicates that if the Ts Payload is after the CP payload. If yes, it means the Tsi and Tsr payload should be with Max port range and address range and protocol is marked as zero.

Return values:

Pointer to Ts IKE payload.

Generate TS payload.

This function generates TSi or TSr payload according to type of next payload. If the next payload is Responder TS, gereate TSi Payload. Otherwise, generate TSr payload.

Parameters:

[in]	*ChildSa*	Pointer to IKEV2_CHILD_SA_SESSION related to this TS payload.
[in]	*NextPayload*	The payload type presented in the NextPayload field of ID Payload header.
[in]	*IsTunnel*	It indicates that if the Ts Payload is after the CP payload. If yes, it means the Tsi and Tsr payload should be with Max port range and address range and protocol is marked as zero.

Return values:

Pointer to Ts IKE payload.

References TRAFFIC_SELECTOR::EndPort, IKEV2_TS::Header, IkePayloadAlloc(), IkePayloadFree(), IKEV2_PAYLOAD_TYPE_TS_INIT, IKEV2_PAYLOAD_TYPE_TS_RSP, IKEV2_TS_ANY_PORT, IKEV2_TS_ANY_PROTOCOL, IKEV2_TS_TYPE_IPV4_ADDR_RANGE, IKEV2_TS_TYPS_IPV6_ADDR_RANGE, TRAFFIC_SELECTOR::IpProtocolId, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, IKE_PAYLOAD::PayloadType, TRAFFIC_SELECTOR::SelecorLen, TRAFFIC_SELECTOR::StartPort, IKEV2_TS::TSNumbers, and TRAFFIC_SELECTOR::TSType.

Referenced by Ikev2AuthCertGenerator(), and Ikev2AuthPskGenerator().

IKEV2_SA_DATA* Ikev2InitializeSaData (IN IKEV2_SESSION_COMMON * *SessionCommon*)

Create and intialize IKEV2_SA_DATA for speicifed IKEV2_SESSION_COMMON.

This function will be only called by the initiator. The responder's IKEV2_SA_DATA will be generated during parsed the initiator packet.

Parameters:

[in]	*SessionCommon*	Pointer to IKEV2_SESSION_COMMON related to.

Return values:

a Pointer to a new IKEV2_SA_DATA or NULL.

References IKE_SA_ATTRIBUTE::Attr, IKEV2_TRANSFORM_DATA::Attribute, IKE_SA_ATTR_UNION::AttrLength, IKE_SA_ATTRIBUTE::AttrType, IkeSessionTypeIkeSa, IKEV2_ATTRIBUTE_TYPE_KEYLEN, IKEV2_CHILD_SA_SESSION_FROM_COMMON, IKEV2_TRANSFORM_ID_AUTH_HMAC_SHA1_96, IKEV2_TRANSFORM_ID_DH_1024MODP, IKEV2_TRANSFORM_ID_ENCR_3DES, IKEV2_TRANSFORM_ID_ENCR_AES_CBC, IKEV2_TRANSFORM_ID_PRF_HMAC_SHA1, IKEV2_TRANSFORM_TYPE_DH, IKEV2_TRANSFORM_TYPE_ENCR, IKEV2_TRANSFORM_TYPE_ESN, IKEV2_TRANSFORM_TYPE_INTEG, IKEV2_TRANSFORM_TYPE_PRF, IPSEC_PROTO_IPSEC_ESP, IPSEC_PROTO_ISAKMP, IpSecGetEncryptKeyLength(), IKEV2_CHILD_SA_SESSION::LocalPeerSpi, IKEV2_SA_DATA::NumProposals, IKEV2_PROPOSAL_DATA::NumTransforms, IKEV2_PROPOSAL_DATA::ProposalIndex, IKEV2_PROPOSAL_DATA::ProtocolId, IKEV2_PROPOSAL_DATA::Spi, IKEV2_TRANSFORM_DATA::TransformId, IKEV2_TRANSFORM_DATA::TransformIndex, and IKEV2_TRANSFORM_DATA::TransformType.

Referenced by Ikev2HandleSa(), Ikev2NegotiateChildSa(), and Ikev2NegotiateSa().

```
BOOLEAN Ikev2IsSupportAlg ( IN UINT16  AlgorithmId,
                            IN UINT8   Type
                          )
```

Check if the Algorithm ID is supported.

Parameters:

[in] *AlgorithmId* The specified Algorithm ID.
[in] *Type* The type used to indicate the Algorithm is for Encrypt or Authentication.

Return values:

TRUE If the Algorithm ID is supported.
FALSE If the Algorithm ID is not supported.

References IKE_AUTH_TYPE, IKE_DH_TYPE, IKE_ENCRYPT_TYPE, IKE_PRF_TYPE, IKEV2_SUPPORT_AUTH_ALGORITHM_NUM, IKEV2_SUPPORT_DH_ALGORITHM_NUM, IKEV2_SUPPORT_ENCRYPT_ALGORITHM_NUM, IKEV2_SUPPORT_PRF_ALGORITHM_NUM, mIkev2AuthAlgorithmList, mIkev2DhGroupAlgorithmList, mIkev2EncryptAlgorithmList, and mIkev2PrfAlgorithmList.

Referenced by Ikev2ParseProposalData().

```
VOID EFIAPI Ikev2LifetimeNotify ( IN EFI_EVENT  Event,
                                  IN VOID *    Context
                                )
```

Callback function for the IKE life time is over.

This function will mark the related IKE SA Session as deleting and trigger a Information negotiation.

Parameters:

[in] *Event* The time out event.
[in] *Context* Pointer to data passed by caller.

Call back function of the IKE life time is over.

This function will mark the related IKE SA Session as deleting and trigger a Information negotiation.

Parameters:

[in] *Event* The signaled Event.
[in] *Context* Pointer to data passed by caller.

References IKEV2_CHILD_SA_SESSION::ByDelete, IKEV2_SA_SESSION::DeleteSaList, IKEV2_CHILD_SA_SESSION::IkeSaSession, IKEV2_SESSION_COMMON::IkeSessionType, IkeSessionTypeIkeSa, IkeStateSaDeleting, IKEV2_CHILD_SA_SESSION_FROM_COMMON, IKEV2_DUMP_STATE, IKEV2_SA_SESSION_FROM_COMMON, IKEV2_SA_SESSION::InitiatorCookie, IKEV2_CHILD_SA_SESSION::LocalPeerSpi, mIkev2Exchange, _IKE_EXCHANGE_INTERFACE::NegotiateInfo, IKEV2_SA_SESSION::ResponderCookie, IKEV2_SA_SESSION::SessionCommon, and IKEV2_SESSION_COMMON::State.

Referenced by Ikev2ChildSaSessionReg(), and Ikev2SaSessionReg().

```
EFI_STATUS Ikev2MatchSpdEntry ( IN EFI_IPSEC_CONFIG_DATA_TYPE  Type,
```

```
                                    IN EFI_IPSEC_CONFIG_SELECTOR *  Selector,
                                    IN VOID *                       Data,
                                    IN UINTN                        SelectorSize,
                                    IN UINTN                        DataSize,
                                    IN VOID *                       Context
                                  )
```

Check if the SPD is related to the input Child SA Session.

This function is the subfunction of Ikev1AssociateSpdEntry(). It is the call back function of IpSecVisitConfigData().

Parameters:

[in]	*Type*	Type of the input Config Selector.
[in]	*Selector*	Pointer to the Configure Selector to be checked.
[in]	*Data*	Pointer to the Configure Selector's Data passed from the caller.
[in]	*SelectorSize*	The buffer size of Selector.
[in]	*DataSize*	The buffer size of the Data.
[in]	*Context*	The data passed from the caller. It is a Child SA Session in this context.

Return values:

EFI_SUCCESS	The SPD Selector is not related to the Child SA Session.
EFI_ABORTED	The SPD Selector is related to the Child SA session and set the ChildSaSession->Spd to point to this SPD Selector.

References EFI_IPSEC_ANY_PORT, EFI_IPSEC_ANY_PROTOCOL, IKE_DEFAULT_PORT, IkeSearchSpdEntry(), IpSecMatchIpAddress(), IKE_UDP_SERVICE::IpVersion, IKEV2_SESSION_COMMON::LocalPeerIp, IKEV2_CHILD_SA_SESSION::LocalPort, IKEV2_CHILD_SA_SESSION::ProtoId, IKEV2_SESSION_COMMON::RemotePeerIp, IKEV2_CHILD_SA_SESSION::RemotePort, IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_CHILD_SA_SESSION::Spd, and IKEV2_SESSION_COMMON::UdpService.

Referenced by Ikev2ChildSaAssociateSpdEntry().

```
VOID Ikev2OnPacketAccepted ( IN IKEV2_SESSION_COMMON *  SessionCommon,
                             IN IKE_PACKET *            IkePacket,
                             IN UINT8                   IkeType
                           )
```

Save some useful payloads after accepting the Packet.

Parameters:

[in]	*SessionCommon*	Pointer to IKEV2_SESSION_COMMON related to the operation.
[in]	*IkePacket*	Pointer to received IkePacet.
[in]	*IkeType*	The type used to indicate it is in IkeSa or ChildSa or Info exchange.

```
EFI_STATUS Ikev2ParserNotifyCookiePayload ( IN IKE_PAYLOAD *          IkeNCookie,
                                            IN OUT IKEV2_SA_SESSION * IkeSaSession
                                          )
```

Parser the Notify Cookie payload.

This function parses the Notify Cookie payload.If the Notify ProtocolId is not IPSEC_PROTO_ISAKMP or if the SpiSize is not zero or if the MessageType is not the COOKIE, return EFI_INVALID_PARAMETER.

Parameters:

[in]	*IkeNCookie*	Pointer to the IKE_PAYLOAD which contians the Notify Cookie payload. the Notify payload.
[in,out]	*IkeSaSession*	Pointer to the relevant IKE SA Session.

Return values:

EFI_SUCCESS	The Notify Cookie Payload is valid.
EFI_INVALID_PARAMETER	The Notify Cookie Payload is invalid.
EFI_OUT_OF_RESOURCE	The required resource can't be allocated.

Parser the Notify Cookie payload.

This function parses the Notify Cookie payload.If the Notify ProtocolId is not IPSEC_PROTO_ISAKMP or if the SpiSize is not zero or if the MessageType is not the COOKIE, return EFI_INVALID_PARAMETER.

Parameters:

[in]	*IkeNCookie*	Pointer to the IKE_PAYLOAD which contians the Notify Cookie payload. the Notify payload.
[in,out]	*IkeSaSession*	Pointer to the relevant IKE SA Session.

Return values:

EFI_SUCCESS	The Notify Cookie Payload is valid.
EFI_INVALID_PARAMETER	The Notify Cookie Payload is invalid.
EFI_OUT_OF_RESOURCE	The required resource can't be allocated.

References IKEV2_NOTIFY::Header, IKEV2_NOTIFICATION_COOKIE, IPSEC_PROTO_ISAKMP, IKEV2_NOTIFY::MessageType, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKEV2_NOTIFY::ProtocolId, and IKEV2_NOTIFY::SpiSize.

Referenced by Ikev2InitPskParser().

```
IKE_PAYLOAD* Ikev2PskGenerateAuthPayload (
    IN IKEV2_SA_SESSION *   IkeSaSession,
    IN IKE_PAYLOAD *        IdPayload,
    IN UINT8                NextPayload,
    IN BOOLEAN              IsVerify
)
```

Generate a Authentication Payload.

This function is used for both Authentication generation and verification. When the IsVerify is TRUE, it create a Auth Data for verification. This function choose the related IKE_SA_INIT Message for Auth data creation according to the IKE Session's type and the value of IsVerify parameter.

Parameters:

[in]	*IkeSaSession*	Pointer to IKEV2_SA_SESSION related to.
[in]	*IdPayload*	Pointer to the ID payload to be used for Authentication payload generation.
[in]	*NextPayload*	The type filled into the Authentication Payload next payload field.

[in] *IsVerify* If it is TURE, the Authentication payload is used for verification.

Returns:

pointer to IKE Authentication payload for pre-shard key method.

Generate a Authentication Payload.

This function is used for both Authentication generation and verification. When the IsVerify is TRUE, it create a Auth Data for verification. This function choose the related IKE_SA_INIT Message for Auth data creation according to the IKE Session's type and the value of IsVerify parameter.

Parameters:

[in] *IkeSaSession* Pointer to IKEV2_SA_SESSION related to.
[in] *IdPayload* Pointer to the ID payload to be used for Authentication payload generation.
[in] *NextPayload* The type filled into the Authentication Payload next payload field.
[in] *IsVerify* If it is TURE, the Authentication payload is used for verification.

Returns:

pointer to IKE Authentication payload for Pre-shared key method.

References IKEV2_AUTH::AuthMethod, CONSTANT_KEY_SIZE, HASH_DATA_FRAGMENT::Data, PRF_DATA_FRAGMENT::Data, HASH_DATA_FRAGMENT::DataSize, PRF_DATA_FRAGMENT::DataSize, IKEV2_AUTH::Header, IkePayloadAlloc(), IkePayloadFree(), IKEV2_AUTH_METHOD_SKMI, IKEV2_PAYLOAD_TYPE_AUTH, IpSecCryptoIoHmac(), IpSecGetHmacDigestLength(), mConstantKey, IKEV2_COMMON_PAYLOAD_HEADER::NextPayload, IKE_PAYLOAD::PayloadBuf, IKEV2_COMMON_PAYLOAD_HEADER::PayloadLength, IKE_PAYLOAD::PayloadSize, and IKE_PAYLOAD::PayloadType.

Referenced by Ikev2AuthPskGenerator(), and Ikev2AuthPskParser().

```
VOID EFIAPI Ikev2ResendNotify ( IN EFI_EVENT  Event,
                                IN VOID *     Context
                              )
```

This function will be called if the TimeOut Event is signaled.

Parameters:

[in] *Event* The signaled Event.
[in] *Context* The data passed by caller.

References IKEV2_SA_SESSION::ChildSaEstablishSessionList, IKEV2_SA_SESSION::ChildSaSessionList, IKEV2_SA_SESSION::DeleteSaList, IKE_MAX_RETRY, IKEV2_CHILD_SA_SESSION::IkeSaSession, IKEV2_SESSION_COMMON::IkeSessionType, IkeSessionTypeIkeSa, IkeStateSaDeleting, _IPSEC_PRIVATE_DATA::Ikev1EstablishedList, IKEV2_CHILD_SA_SESSION_BY_IKE_SA, IKEV2_CHILD_SA_SESSION_FROM_COMMON, IKEV2_DELET_CHILDSA_LIST, IKEV2_ESTABLISHING_CHILDSA_LIST, IKEV2_SA_SESSION_FROM_COMMON, Ikev2ChildSaSessionFree(), Ikev2ChildSaSessionRemove(), Ikev2ChildSaSilentDelete(), _IPSEC_PRIVATE_DATA::Ikev2EstablishedList, Ikev2SaSessionFree(), Ikev2SaSessionRemove(), Ikev2SendIkePacket(), _IPSEC_PRIVATE_DATA::Ikev2SessionList, _IPSEC_PRIVATE_DATA::IpSec, IPSEC_STATUS_DISABLED, IPSECCONFIG_STATUS_NAME, _IPSEC_PRIVATE_DATA::IsIPsecDisabling, IKEV2_SESSION_COMMON::LastSentPacket, IKEV2_CHILD_SA_SESSION::LocalPeerSpi, IKEV2_SESSION_COMMON::Private, IKEV2_SESSION_COMMON::RemotePeerIp, IKEV2_SESSION_COMMON::RetryCount,

IKEV2_CHILD_SA_SESSION::SessionCommon, IKEV2_SA_SESSION::SessionCommon, IKEV2_SESSION_COMMON::State, and IKEV2_SESSION_COMMON::UdpService.

Referenced by Ikev2ChildSaSessionAlloc(), and Ikev2SaSessionAlloc().

```
EFI_STATUS Ikev2SaGenerateKey ( IN UINT8                    HashAlgId,
                                IN UINT8 *                  HashKey,
                                IN UINTN                    HashKeyLength,
                                IN OUT UINT8 *              OutputKey,
                                IN UINTN                    OutputKeyLength,
                                IN PRF_DATA_FRAGMENT *      Fragments,
                                IN UINTN                    NumFragments
                              )
```

Generate Key buffer from fragments.

If the digest length of specified HashAlgId is larger than or equal with the required output key length, derive the key directly. Otherwise, Key Material needs to be PRF-based concatenation according to 2.13 of RFC 4306: prf+ (K,S) = T1 | T2 | T3 | T4 | ..., T1 = prf (K, S | 0x01), T2 = prf (K, T1 | S | 0x02), T3 = prf (K, T2 | S | 0x03),T4 = prf (K, T3 | S | 0x04) then derive the key from this key material.

Parameters:

[in]	*HashAlgId*	The Hash Algorithm ID used to generate key.
[in]	*HashKey*	Pointer to a key buffer which contains hash key.
[in]	*HashKeyLength*	The length of HashKey in bytes.
[in,out]	*OutputKey*	Pointer to buffer which is used to receive the output key.
[in]	*OutputKeyLength*	The length of OutPutKey buffer.
[in]	*Fragments*	Pointer to the data to be used to generate key.
[in]	*NumFragments*	The numbers of the Fragement.

Return values:

EFI_SUCCESS	The operation complete successfully.
EFI_INVALID_PARAMETER	If NumFragments is zero.
EFI_OUT_OF_RESOURCES	If the required resource can't be allocated.
Others	The operation is failed.

References PRF_DATA_FRAGMENT::Data, PRF_DATA_FRAGMENT::DataSize, IpSecCryptoIoHmac(), and IpSecGetHmacDigestLength().

Referenced by Ikev2GenerateChildSaKeys(), and Ikev2GenerateSaKeys().

```
BOOLEAN Ikev2SaParseSaPayload ( IN OUT IKEV2_SA_SESSION *   IkeSaSession,
                                IN IKE_PAYLOAD *            SaPayload,
                                IN UINT8                    Type
                              )
```

Parse the received Initial Exchange Packet.

This function parse the SA Payload and Key Payload to find out the cryptographic suite for the further IKE negotiation and fill it into the IKE SA Session's CommonSession->SaParams.

Parameters:

[in,out]	*IkeSaSession*	Pointer to related IKEV2_SA_SESSION.
[in]	*SaPayload*	The received packet.
[in]	*Type*	The received packet IKE header flag.

Return values:

TRUE	If the SA proposal in Packet is acceptable.
FALSE	If the SA proposal in Packet is not acceptable.

References IKE_HEADER_FLAGS_INIT, IKE_HEADER_FLAGS_RESPOND, Ikev2ParseProposalData(), IPSEC_PROTO_ISAKMP, IKEV2_PROPOSAL_DATA::NumTransforms, and IKEV2_PROPOSAL_DATA::ProtocolId.

Referenced by Ikev2InitPskParser().

IKEV2_SA_SESSION* Ikev2SaSessionAlloc (IN IPSEC_PRIVATE_DATA * *Private*,
IN IKE_UDP_SERVICE * *UdpService*
)

Allocate buffer for IKEV2_SA_SESSION and initialize it.

Parameters:

[in]	*Private*	Pointer to IPSEC_PRIVATE_DATA.
[in]	*UdpService*	Pointer to IKE_UDP_SERVICE related to this IKE SA Session.

Returns:

Pointer to IKEV2_SA_SESSION.

Allocate buffer for IKEV2_SA_SESSION and initialize it.

Parameters:

[in]	*Private*	Pointer to IPSEC_PRIVATE_DATA.
[in]	*UdpService*	Pointer to IKE_UDP_SERVICE related to this IKE SA Session.

Returns:

Pointer to IKEV2_SA_SESSION or NULL.

References IKEV2_SESSION_COMMON::AfterEncodePayload, IKEV2_SESSION_COMMON::BeforeDecodePayload, IKEV2_SA_SESSION::ChildSaEstablishSessionList, IKEV2_SA_SESSION::ChildSaSessionList, IKEV2_SA_SESSION::DeleteSaList, IkeGenerateCookie(), IKEV2_SESSION_COMMON::IkeSessionType, IkeSessionTypeIkeSa, IKEV2_SA_SESSION_SIGNATURE, Ikev2ResendNotify(), IKEV2_SESSION_COMMON::IkeVer, IKEV2_SA_SESSION::InfoMIDList, IKEV2_SA_SESSION::InitiatorCookie, IKEV2_SA_SESSION::MessageId, IKEV2_SA_SESSION::NCookie, IKEV2_SESSION_COMMON::Private, IKEV2_SA_SESSION::ResponderCookie, IKEV2_SA_SESSION::SessionCommon, IKEV2_SA_SESSION::Signature, IKEV2_SESSION_COMMON::TimeoutEvent, and IKEV2_SESSION_COMMON::UdpService.

Referenced by Ikev2HandleSa(), and Ikev2NegotiateSa().

VOID Ikev2SaSessionCommonFree (IN IKEV2_SESSION_COMMON * *SessionCommon*)

Free specified Seession Common. The session common would belong to a IKE SA or a Child SA.

Parameters:

[in] *SessionCommon* Pointer to a Session Common.

References IkePacketFree().

Referenced by Ikev2ChildSaSessionFree(), and Ikev2SaSessionFree().

VOID Ikev2SaSessionFree (IN IKEV2_SA_SESSION * *IkeSaSession*)

Free specified IKEV2 SA Session.

Parameters:

[in] *IkeSaSession* Pointer to IKEV2_SA_SESSION to be freed.

References IKEV2_SESSION_KEYS::DhBuffer, IKEV2_CHILD_SA_SESSION::IkeSaSession, IKEV2_CHILD_SA_SESSION_BY_IKE_SA, Ikev2ChildSaSessionFree(), Ikev2ChildSaSilentDelete(), Ikev2DhBufferFree(), Ikev2SaSessionCommonFree(), IKEV2_CHILD_SA_SESSION::LocalPeerSpi, IKEV2_SESSION_KEYS::SkAiKey, IKEV2_SESSION_KEYS::SkArKey, IKEV2_SESSION_KEYS::SkdKey, IKEV2_SESSION_KEYS::SkEiKey, IKEV2_SESSION_KEYS::SkErKey, IKEV2_SESSION_KEYS::SkPiKey, and IKEV2_SESSION_KEYS::SkPrKey.

Referenced by IkeDeleteAllSas(), Ikev2HandleSa(), Ikev2InfoParser(), Ikev2NegotiateSa(), Ikev2OnPacketSent(), Ikev2ResendNotify(), Ikev2SaSessionReg(), and IpSecStop().

VOID Ikev2SaSessionIncreaseMessageId (IN IKEV2_SA_SESSION * *IkeSaSession*)

Increase the MessageID in IkeSaSession.

Parameters:

[in] *IkeSaSession* Pointer to a specified IKEV2_SA_SESSION.

Referenced by Ikev2InfoGenerator().

VOID Ikev2SaSessionInsert (IN LIST_ENTRY * *SaSessionList*,
IN IKEV2_SA_SESSION * *IkeSaSession*,
IN EFI_IP_ADDRESS * *RemotePeerIp*
)

Insert a IKE_SA_SESSION into IkeSaSession list. The IkeSaSession list is either Private->Ikev2SaSession list or Private->Ikev2EstablishedList list.

Parameters:

[in] *SaSessionList* Pointer to list to be inserted into.

[in] *IkeSaSession* Pointer to IKEV2_SA_SESSION to be inserted.

[in] *RemotePeerIp* Pointer to EFI_IP_ADDRESSS to indicate the unique IKEV2_SA_SESSION.

References Ikev2SaSessionRemove().

Referenced by Ikev2HandleSa(), Ikev2NegotiateSa(), and Ikev2SaSessionReg().

IKEV2_SA_SESSION* Ikev2SaSessionLookup (IN LIST_ENTRY * *SaSessionList*,
IN EFI_IP_ADDRESS * *RemotePeerIp*
)

Find a IKEV2_SA_SESSION by the remote peer IP.

Parameters:

[in] *SaSessionList* SaSession List to be searched.
[in] *RemotePeerIp* Pointer to specified IP address.

Returns:

Pointer to IKEV2_SA_SESSION if find one or NULL.

References IKEV2_SA_SESSION_BY_SESSION, IKEV2_SESSION_COMMON::RemotePeerIp, and IKEV2_SA_SESSION::SessionCommon.

Referenced by IkeNegotiate(), Ikev2HandleChildSa(), Ikev2HandleInfo(), Ikev2HandleSa(), Ikev2NegotiateSa(), and Ikev2OnPacketSent().

EFI_STATUS Ikev2SaSessionOnDeleting (IN IKEV2_SA_SESSION * *IkeSaSession*)

Marking a SA session as on deleting.

Parameters:

[in] *IkeSaSession* Pointer to IKEV2_SA_SESSION.

Return values:

EFI_SUCCESS Find the related SA session and marked it.

VOID Ikev2SaSessionReg (IN IKEV2_SA_SESSION * *IkeSaSession*,
IN IPSEC_PRIVATE_DATA * *Private*
)

Register Establish IKEv2 SA into Private->Ikev2EstablishedList.

Parameters:

[in] *IkeSaSession* Pointer to IKEV2_SA_SESSION to be registered.
[in] *Private* Pointer to IPSEC_PRAVATE_DATA.

Register the established IKEv2 SA into Private->Ikev2EstablishedList. If there is IKEV2_SA_SESSION with same remote peer IP, remove the old one then register the new one.

Parameters:

[in] *IkeSaSession* Pointer to IKEV2_SA_SESSION to be registered.
[in] *Private* Pointer to IPSEC_PRAVATE_DATA.

References IKE_SA_DEFAULT_LIFETIME, Ikev2LifetimeNotify(), Ikev2SaSessionFree(), Ikev2SaSessionInsert(), Ikev2SaSessionRemove(), Ikev2SessionCommonRefresh(), IKEV2_SESSION_COMMON::RemotePeerIp, and IKEV2_SESSION_COMMON::TimeoutEvent.

Referenced by Ikev2HandleSa().

```
IKEV2_SA_SESSION* Ikev2SaSessionRemove ( IN LIST_ENTRY *        SaSessionList,
                                         IN EFI_IP_ADDRESS *    RemotePeerIp
                                       )
```

Remove the SA Session by Remote Peer IP.

Parameters:

[in] *SaSessionList* Pointer to list to be searched.
[in] *RemotePeerIp* Pointer to EFI_IP_ADDRESS to use for SA Session search.

Return values:

Pointer to IKEV2_SA_SESSION with the specified remote IP address.

Remove the SA Session by Remote Peer IP.

Parameters:

[in] *SaSessionList* Pointer to list to be searched.
[in] *RemotePeerIp* Pointer to EFI_IP_ADDRESS to use for SA Session search.

Return values:

Pointer to IKEV2_SA_SESSION with the specified remote IP address or NULL.

References IKEV2_SA_SESSION_BY_SESSION, IKEV2_SESSION_COMMON::RemotePeerIp, and IKEV2_SA_SESSION::SessionCommon.

Referenced by Ikev2HandleSa(), Ikev2ResendNotify(), Ikev2SaSessionInsert(), and Ikev2SaSessionReg().

```
EFI_STATUS Ikev2SendIkePacket ( IN IKE_UDP_SERVICE *   IkeUdpService,
                                IN UINT8 *             SessionCommon,
                                IN IKE_PACKET *        IkePacket,
                                IN UINTN               IkeType
                              )
```

Send out IKEV2 packet.

Parameters:

[in] *IkeUdpService* Pointer to IKE_UDP_SERVICE used to send the IKE packet.
[in] *SessionCommon* Pointer to IKEV1_SESSION_COMMON related to the IKE packet.
[in] *IkePacket* Pointer to IKE_PACKET to be sent out.
[in] *IkeType* The type of IKE to point what's kind of the IKE packet is to be sent out. IKE_SA_TYPE, IKE_INFO_TYPE and IKE_CHILD_TYPE are supportted.

Return values:

EFI_SUCCESS The operation complete successfully.
Otherwise The operation is failed.

Send out IKEV2 packet.

Parameters:

[in] *IkeUdpService* Pointer to IKE_UDP_SERVICE used to send the IKE packet.
[in] *SessionCommon* Pointer to IKEV1_SESSION_COMMON related to the IKE packet.
[in] *IkePacket* Pointer to IKE_PACKET to be sent out.
[in] *IkeType*

The type of IKE to point what's kind of the IKE packet is to be sent out. IKE_SA_TYPE, IKE_INFO_TYPE and IKE_CHILD_TYPE are supportted.

Return values:

EFI_SUCCESS The operation complete successfully.
Otherwise The operation is failed.

References IKE_DEFAULT_PORT, IKE_DEFAULT_TIMEOUT_INTERVAL, IKE_HEADER_FLAGS_INIT, IKE_PACKET_REF, IkeNetbufFromPacket(), IkePacketFree(), Ikev2OnPacketSent(), IPSEC_DUMP_PACKET, IKEV2_SESSION_COMMON::LastSentPacket, IKEV2_SESSION_COMMON::LocalPeerIp, IKEV2_SESSION_COMMON::RemotePeerIp, IKEV2_SESSION_COMMON::TimeoutEvent, and IKEV2_SESSION_COMMON::TimeoutInterval.

Referenced by Ikev2HandleChildSa(), Ikev2HandleSa(), Ikev2InfoParser(), Ikev2NegotiateChildSa(), Ikev2NegotiateInfo(), Ikev2NegotiateSa(), and Ikev2ResendNotify().

VOID Ikev2SessionCommonRefresh (IN IKEV2_SESSION_COMMON * *SessionCommon*)

After IKE/Child SA is estiblished, close the time event and free sent packet.

Parameters:

`[in]` *SessionCommon* Pointer to a Session Common.

References IkePacketFree().

Referenced by Ikev2ChildSaSessionReg(), and Ikev2SaSessionReg().

VOID Ikev2StoreSaData (IN IKEV2_CHILD_SA_SESSION * *ChildSaSession*)

Store the SA into SAD.

Parameters:

`[in]` *ChildSaSession* Pointer to IKEV2_CHILD_SA_SESSION.

References EfiIpSecConfigSetData(), _IPSEC_PRIVATE_DATA::IpSecConfig, and IKEV2_SESSION_COMMON::Private.

Referenced by Ikev2ChildSaSessionReg().

BOOLEAN Ikev2ValidateHeader (IN IKEV2_SA_SESSION * *IkeSaSession*,
IN IKE_HEADER * *IkeHdr*
)

Validate the IKE header of received IKE packet.

Parameters:

`[in]` *IkeSaSession* Pointer to IKEV2_SA_SESSION related to this IKE packet.
`[in]` *IkeHdr* Pointer to IKE header of received IKE packet.

Return values:

TRUE If the IKE header is valid.
FALSE If the IKE header is invalid.

References IKE_HEADER_FLAGS_INIT, IKE_HEADER_FLAGS_RESPOND, IkeStateAuth, IkeStateInit, IKEV2_EXCHANGE_TYPE_CREATE_CHILD, and IKEV2_EXCHANGE_TYPE_INFO.

Referenced by Ikev2HandleChildSa(), Ikev2HandleInfo(), and Ikev2HandleSa().

Variable Documentation

IKE_ALG_GUID_INFO mIPsecEncrAlgInfo[]

NetworkPkg/IScsiDxe/ComponentName.c File Reference

Functions

EFI_STATUS EFIAPI IScsiComponentNameGetDriverName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN CHAR8 *Language, OUT CHAR16 **DriverName)

EFI_STATUS UpdateName (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *IScsiExtScsiPassThru, IN BOOLEAN Ipv6Flag)

EFI_STATUS EFIAPI IScsiComponentNameGetControllerName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE ChildHandle, IN CHAR8 *Language, OUT CHAR16 **ControllerName)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gIScsiComponentName

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gIScsiComponentName2

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mIScsiDriverNameTable []

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE * gIScsiControllerNameTable = NULL

Detailed Description

UEFI Component Name(2) protocol implementation for iSCSI.

Copyright (c) 2004 - 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI IScsiComponentNameGetControllerName	(	IN EFI_COMPONENT_NAME_PROTOCOL *	*This*,
		IN EFI_HANDLE	*ControllerHandle*,
		IN EFI_HANDLE	*ChildHandle*,
		IN CHAR8 *	*Language*,

OUT CHAR16 ** *ControllerName*
)

Retrieves a Unicode string that is the user readable name of the controller that is being managed by a driver.

This function retrieves the user readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

`[in]`	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
`[in]`	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
`[in]`	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
`[in]`	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
`[out]`	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language, from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL, and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gIScsiComponentName, gIScsiControllerNameTable, gIScsiV4PrivateGuid, gIScsiV6PrivateGuid, and UpdateName().

EFI_STATUS EFIAPI IScsiComponentNameGetDriverName
(IN EFI_COMPONENT_NAME_PROTOCOL * *This*,
IN CHAR8 * *Language*,

OUT CHAR16 ** *DriverName*
)

Retrieves a Unicode string that is the user readable name of the driver.

This function retrieves the user readable name of a driver in the form of a Unicode string. If the driver specified by This has a user readable name in the language specified by Language, then a pointer to the driver name is returned in DriverName, and EFI_SUCCESS is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*DriverName*	A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by This in the language specified by Language.

Return values:

EFI_SUCCESS	The Unicode string for the Driver specified by This and the language specified by Language was returned in DriverName.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	DriverName is NULL.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gIScsiComponentName, and mIScsiDriverNameTable.

EFI_STATUS UpdateName (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL * *IScsiExtScsiPassThru*,
IN BOOLEAN *Ipv6Flag*
)

Update the component name for the iSCSI instance.

Parameters:

[in]	*IScsiExtScsiPassThru*	A pointer to the EFI_EXT_SCSI_PASS_THRU_PROTOCOL instance.
[in]	*Ipv6Flag*	TRUE if IP6 network stack is used.

Return values:

EFI_SUCCESS	Update the ControllerNameTable of this instance successfully.
EFI_INVALID_PARAMETER	The input parameter is invalid.
EFI_UNSUPPORTED	Can't get the corresponding NIC info from the Controller handle.

References _ISCSI_SESSION::ConfigData, gIScsiComponentName, gIScsiComponentName2, gIScsiControllerNameTable, ISCSI_DRIVER_DATA_FROM_EXT_SCSI_PASS_THRU, _ISCSI_ATTEMPT_CONFIG_NVDATA::NicIndex, and _ISCSI_DRIVER_DATA::Session.

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gIScsiComponentName

Initial value:

```
{
 IScsiComponentNameGetDriverName,
 IScsiComponentNameGetControllerName,
 "eng"
}
```

Referenced by IScsiComponentNameGetControllerName(), IScsiComponentNameGetDriverName(), IScsiDriverEntryPoint(), and UpdateName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gIScsiComponentName2

Initial value:

```
{
 (EFI_COMPONENT_NAME2_GET_DRIVER_NAME) IScsiComponentNameGetDriverName,
 (EFI_COMPONENT_NAME2_GET_CONTROLLER_NAME) IScsiComponentNameGetControllerName,
 "en"
}
```

Referenced by IScsiDriverEntryPoint(), and UpdateName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE* gIScsiControllerNameTable = NULL

Referenced by IScsiComponentNameGetControllerName(), IScsiUnload(), and UpdateName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mIScsiDriverNameTable[]

Initial value:

```
{
 {
   "eng;en",
   L"iSCSI Driver"
 },
 {
   NULL,
   NULL
 }
}
```

Referenced by IScsiComponentNameGetDriverName().

- Globals

NetworkPkg/IScsiDxe/IScsiAuthenticationInfo.c File Reference

Functions

EFI_STATUS EFIAPI IScsiGetAuthenticationInfo (IN EFI_AUTHENTICATION_INFO_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, OUT VOID **Buffer)

EFI_STATUS EFIAPI IScsiSetAuthenticationInfo (IN EFI_AUTHENTICATION_INFO_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN VOID *Buffer)

Variables

EFI_AUTHENTICATION_INFO_PROTOCOL gIScsiAuthenticationInfo

Detailed Description

Implementation for EFI_AUTHENTICATION_INFO_PROTOCOL. Currently it is a dummy support.

Copyright (c) 2009 - 2011, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI IScsiGetAuthenticationInfo	(IN EFI_AUTHENTICATION_INFO_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	OUT VOID **	*Buffer*
	)	

Retrieves the authentication information associated with a particular controller handle.

Parameters:

[in]	*This*	Pointer to the EFI_AUTHENTICATION_INFO_PROTOCOL.
[in]	*ControllerHandle*	Handle to the Controller.
[out]	*Buffer*	Pointer to the authentication information. This function is responsible for allocating the buffer and it is the caller's responsibility to free buffer when the caller is finished with buffer.

Return values:

EFI_DEVICE_ERROR The authentication information could not be retrieved due to a hardware error.

EFI_STATUS EFIAPI IScsiSetAuthenticationInfo	(IN EFI_AUTHENTICATION_INFO_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN VOID *	*Buffer*
	)	

Set the authentication information for a given controller handle.

Parameters:

[in]	*This*	Pointer to the EFI_AUTHENTICATION_INFO_PROTOCOL.
[in]	*ControllerHandle*	Handle to the Controller.
[in]	*Buffer*	Pointer to the authentication information.

Return values:

EFI_UNSUPPORTED If the platform policies do not allow setting of the authentication information.

Variable Documentation

EFI_AUTHENTICATION_INFO_PROTOCOL gIScsiAuthenticationInfo

Initial value:

```
{
  IScsiGetAuthenticationInfo,
  IScsiSetAuthenticationInfo
}
```

Referenced by IScsiDriverEntryPoint(), and IScsiUnload().

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IScsiDxe/IScsiCHAP.c File Reference

Functions

EFI_STATUS IScsiCHAPCalculateResponse (IN UINT32 ChapIdentifier, IN CHAR8 *ChapSecret, IN UINT32 SecretLength, IN UINT8 *ChapChallenge, IN UINT32 ChallengeLength, OUT UINT8 *ChapResponse)
EFI_STATUS IScsiCHAPAuthTarget (IN ISCSI_CHAP_AUTH_DATA *AuthData, IN UINT8 *TargetResponse)
EFI_STATUS IScsiCHAPOnRspReceived (IN ISCSI_CONNECTION *Conn)
EFI_STATUS IScsiCHAPToSendReq (IN ISCSI_CONNECTION *Conn, IN OUT NET_BUF *Pdu)

Detailed Description

This file is for Challenge-Handshake Authentication Protocol (CHAP) Configuration.

Copyright (c) 2004 - 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS IScsiCHAPAuthTarget (IN ISCSI_CHAP_AUTH_DATA * *AuthData*,
IN UINT8 * *TargetResponse*
)

The initator checks the CHAP response replied by target against its own calculation of the expected hash value.

Parameters:

[in]	*AuthData*	iSCSI CHAP authentication data.
[in]	*TargetResponse*	The response from target.

Return values:

EFI_SUCCESS	The response from target passed authentication.
EFI_SECURITY_VIOLATION	The response from target was not expected value.
Others	Other errors as indicated.

References ISCSI_CHAP_RSP_LEN, and IScsiCHAPCalculateResponse().

Referenced by IScsiCHAPOnRspReceived().

EFI_STATUS IScsiCHAPCalculateResponse (IN UINT32 *ChapIdentifier*,

```
                                    IN CHAR8 *     ChapSecret,
                                    IN UINT32      SecretLength,
                                    IN UINT8 *     ChapChallenge,
                                    IN UINT32      ChallengeLength,
                                    OUT UINT8 *    ChapResponse
                                  )
```

Initator calculates its own expected hash value.

Parameters:

[in]	*ChapIdentifier*	iSCSI CHAP identifier sent by authenticator.
[in]	*ChapSecret*	iSCSI CHAP secret of the authenticator.
[in]	*SecretLength*	The length of iSCSI CHAP secret.
[in]	*ChapChallenge*	The challenge message sent by authenticator.
[in]	*ChallengeLength*	The length of iSCSI CHAP challenge message.
[out]	*ChapResponse*	The calculation of the expected hash value.

Return values:

EFI_SUCCESS	The expected hash value was calculatedly successfully.
EFI_PROTOCOL_ERROR	The length of the secret should be at least the length of the hash value for the hashing algorithm chosen.
EFI_PROTOCOL_ERROR	MD5 hash operation fail.
EFI_OUT_OF_RESOURCES	Fail to allocate resource to complete MD5.

References ISCSI_CHAP_SECRET_MIN_LEN.

Referenced by IScsiCHAPAuthTarget(), and IScsiCHAPOnRspReceived().

EFI_STATUS IScsiCHAPOnRspReceived (IN ISCSI_CONNECTION * *Conn*)

This function checks the received iSCSI Login Response during the security negotiation stage.

Parameters:

[in] *Conn* The iSCSI connection.

Return values:

EFI_SUCCESS	The Login Response passed the CHAP validation.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_PROTOCOL_ERROR	Some kind of protocol error occurred.
Others	Other errors as indicated.

References _ISCSI_CHAP_AUTH_DATA::AuthConfig, _ISCSI_SESSION::AuthData, _ISCSI_SESSION::AuthType, _ISCSI_SESSION::CHAP, _ISCSI_CHAP_AUTH_DATA::CHAPResponse, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPSecret, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPType, _ISCSI_CHAP_AUTH_DATA::InChallenge, _ISCSI_CHAP_AUTH_DATA::InChallengeLength, _ISCSI_CHAP_AUTH_DATA::InIdentifier, ISCSI_AUTH_INITIAL, ISCSI_AUTH_METHOD_CHAP, ISCSI_AUTH_TYPE_CHAP, ISCSI_AUTH_TYPE_NONE, ISCSI_CHAP_ALGORITHM_MD5, ISCSI_CHAP_AUTH_MAX_LEN, ISCSI_CHAP_MUTUAL, ISCSI_CHAP_RSP_LEN, ISCSI_CHAP_STEP_FOUR, ISCSI_CHAP_STEP_ONE, ISCSI_CHAP_STEP_THREE, ISCSI_CHAP_STEP_TWO, ISCSI_KEY_AUTH_METHOD, ISCSI_KEY_CHAP_ALGORITHM, ISCSI_KEY_CHAP_CHALLENGE, ISCSI_KEY_CHAP_IDENTIFIER, ISCSI_KEY_CHAP_NAME, ISCSI_KEY_CHAP_RESPONSE, ISCSI_KEY_TARGET_PORTAL_GROUP_TAG, ISCSI_KEY_VALUE_NONE,

ISCSI_SECURITY_NEGOTIATION, IScsiBuildKeyValueList(), IScsiCHAPAuthTarget(), IScsiCHAPCalculateResponse(), IScsiFreeKeyValueList(), IScsiGetValueByKeyFromList(), IScsiHexToBin(), IScsiNetNtoi(), and _ISCSI_SESSION::TargetPortalGroupTag.

Referenced by IScsiProcessLoginRsp().

```
EFI_STATUS IScsiCHAPToSendReq ( IN ISCSI_CONNECTION *   Conn,
                                IN OUT NET_BUF *        Pdu
                              )
```

This function fills the CHAP authentication information into the login PDU during the security negotiation stage in the iSCSI connection login.

Parameters:

`[in]`	*Conn*	The iSCSI connection.
`[in,out]`	*Pdu*	The PDU to send out.

Return values:

EFI_SUCCESS	All check passed and the phase-related CHAP authentication info is filled into the iSCSI PDU.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_PROTOCOL_ERROR	Some kind of protocol error occurred.

References _ISCSI_CHAP_AUTH_DATA::AuthConfig, _ISCSI_SESSION::AuthData, _ISCSI_SESSION::AuthType, _ISCSI_SESSION::CHAP, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPName, _ISCSI_CHAP_AUTH_DATA::CHAPResponse, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPType, _ISCSI_SESSION::ConfigData, ISCSI_PRIVATE_DATA::InitiatorName, ISCSI_AUTH_INITIAL, ISCSI_AUTH_METHOD_CHAP, ISCSI_AUTH_TYPE_NONE, ISCSI_CHAP_ALGORITHM_MD5, ISCSI_CHAP_MUTUAL, ISCSI_CHAP_RSP_LEN, ISCSI_CHAP_STEP_FOUR, ISCSI_CHAP_STEP_ONE, ISCSI_CHAP_STEP_THREE, ISCSI_CHAP_STEP_TWO, ISCSI_KEY_AUTH_METHOD, ISCSI_KEY_CHAP_ALGORITHM, ISCSI_KEY_CHAP_CHALLENGE, ISCSI_KEY_CHAP_IDENTIFIER, ISCSI_KEY_CHAP_NAME, ISCSI_KEY_CHAP_RESPONSE, ISCSI_KEY_INITIATOR_NAME, ISCSI_KEY_SESSION_TYPE, ISCSI_KEY_TARGET_NAME, ISCSI_KEY_VALUE_NONE, ISCSI_LOGIN_REQ_PDU_FLAG_TRANSIT, ISCSI_SECURITY_NEGOTIATION, ISCSI_SET_FLAG, IScsiAddKeyValuePair(), IScsiBinToHex(), IScsiGenRandom(), mPrivate, _ISCSI_CHAP_AUTH_DATA::OutChallenge, _ISCSI_CHAP_AUTH_DATA::OutChallengeLength, _ISCSI_CHAP_AUTH_DATA::OutIdentifier, _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData, and _ISCSI_SESSION_CONFIG_NVDATA::TargetName.

Referenced by IScsiPrepareLoginReq().

NetworkPkg/IScsiDxe/IScsiCHAP.h File Reference

Data Structures

struct _ISCSI_CHAP_AUTH_CONFIG_N
struct _ISCSI_CHAP_AUTH_DATA

Defines

"memItemRight" valign="bottom">ISCSI_AUTH_METHOD_CHAP "CHAP"
#define ISCSI_KEY_CHAP_ALGORITHM
#define ISCSI_KEY_CHAP_IDENTIFIER
#define ISCSI_KEY_CHAP_CHALLENGE
#define ISCSI_KEY_CHAP_NAME "CHA
#define ISCSI_KEY_CHAP_RESPONSE "
#define ISCSI_CHAP_ALGORITHM_MD5
#define ISCSI_CHAP_AUTH_MAX_LEN
#define ISCSI_CHAP_RSP_LEN 16
#define ISCSI_CHAP_STEP_ONE 1
#define ISCSI_CHAP_STEP_TWO 2
#define ISCSI_CHAP_STEP_THREE 3
#define ISCSI_CHAP_STEP_FOUR 4

Typedefs

typedef struct _ISCSI_CHAP_AUTH_CONFIG_NVDATA ISCSI_CHAP_AUTH_CONFIG_NV
typedef struct _ISCSI_CHAP_AUTH_DATA ISCSI_CHAP_AUTH_DATA

Functions

EFI_STATUS IScsiCHAPOnRspReceived (IN ISCSI_CONNECTION *Conn)
EFI_STATUS IScsiCHAPToSendReq (IN ISCSI_CONNECTION *Conn, IN O NET_BUF *Pdu)

Detailed Description

The header file of CHAP configuration.

Copyright (c) 2004 - 2011, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by IScsiCHAPOnRspReceived(), and IScsiCHAPToSendReq().

#define ISCSI_CHAP_ALGORITHM_MD5 5

Referenced by IScsiCHAPOnRspReceived(), and IScsiCHAPToSendReq().

#define ISCSI_CHAP_AUTH_MAX_LEN 1024

Referenced by IScsiCHAPOnRspReceived().

#define ISCSI_CHAP_RSP_LEN 16
MD5_HASHSIZE

Referenced by IScsiCHAPAuthTarget(), IScsiCHAPOnRspReceived(), and IScsiCHAPToSendReq().

#define ISCSI_CHAP_STEP_FOUR 4

Referenced by IScsiCHAPOnRspReceived(), and IScsiCHAPToSendReq().

#define ISCSI_CHAP_STEP_ONE 1

Referenced by IScsiCHAPOnRspReceived(), and IScsiCHAPToSendReq().

#define ISCSI_CHAP_STEP_THREE 3

Referenced by IScsiCHAPOnRspReceived(), and IScsiCHAPToSendReq().

#define ISCSI_CHAP_STEP_TWO 2

Referenced by IScsiCHAPOnRspReceived(), and IScsiCHAPToSendReq().

#define ISCSI_KEY_CHAP_ALGORITHM "CHAP_A"

Referenced by IScsiCHAPOnRspReceived(), and IScsiCHAPToSendReq().

#define ISCSI_KEY_CHAP_CHALLENGE "CHAP_C"

Referenced by IScsiCHAPOnRspReceived(), and IScsiCHAPToSendReq().

#define ISCSI_KEY_CHAP_IDENTIFIER "CHAP_I"

Referenced by IScsiCHAPOnRspReceived(), and IScsiCHAPToSendReq().

#define ISCSI_KEY_CHAP_NAME "CHAP_N"

Referenced by IScsiCHAPOnRspReceived(), and IScsiCHAPToSendReq().

#define ISCSI_KEY_CHAP_RESPONSE "CHAP_R"

Referenced by IScsiCHAPOnRspReceived(), and IScsiCHAPToSendReq().

Typedef Documentation

typedef struct _ISCSI_CHAP_AUTH_CONFIG_NVDATA ISCSI_CHAP_AUTH_CONFIG_NVDATA

typedef struct _ISCSI_CHAP_AUTH_DATA ISCSI_CHAP_AUTH_DATA
ISCSI CHAP Authentication Data

Function Documentation

EFI_STATUS IScsiCHAPOnRspReceived (IN ISCSI_CONNECTION * *Conn*)
This function checks the received iSCSI Login Response during the security negotiation stage.

Parameters:

[in] *Conn* The iSCSI connection.

Return values:

EFI_SUCCESS	The Login Response passed the CHAP validation.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_PROTOCOL_ERROR	Some kind of protocol error occurred.
Others	Other errors as indicated.

References _ISCSI_CHAP_AUTH_DATA::AuthConfig, _ISCSI_SESSION::AuthData, _ISCSI_SESSION::AuthType, _ISCSI_SESSION::CHAP, _ISCSI_CHAP_AUTH_DATA::CHAPResponse, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPSecret, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPType, _ISCSI_CHAP_AUTH_DATA::InChallenge, _ISCSI_CHAP_AUTH_DATA::InChallengeLength, _ISCSI_CHAP_AUTH_DATA::InIdentifier, ISCSI_AUTH_INITIAL, ISCSI_AUTH_METHOD_CHAP, ISCSI_AUTH_TYPE_CHAP, ISCSI_AUTH_TYPE_NONE, ISCSI_CHAP_ALGORITHM_MD5, ISCSI_CHAP_AUTH_MAX_LEN, ISCSI_CHAP_MUTUAL, ISCSI_CHAP_RSP_LEN, ISCSI_CHAP_STEP_FOUR, ISCSI_CHAP_STEP_ONE, ISCSI_CHAP_STEP_THREE, ISCSI_CHAP_STEP_TWO, ISCSI_KEY_AUTH_METHOD, ISCSI_KEY_CHAP_ALGORITHM, ISCSI_KEY_CHAP_CHALLENGE, ISCSI_KEY_CHAP_IDENTIFIER, ISCSI_KEY_CHAP_NAME, ISCSI_KEY_CHAP_RESPONSE, ISCSI_KEY_TARGET_PORTAL_GROUP_TAG, ISCSI_KEY_VALUE_NONE, ISCSI_SECURITY_NEGOTIATION, IScsiBuildKeyValueList(), IScsiCHAPAuthTarget(),

IScsiCHAPCalculateResponse(), IScsiFreeKeyValueList(), IScsiGetValueByKeyFromList(), IScsiHexToBin(), IScsiNetNtoi(), and _ISCSI_SESSION::TargetPortalGroupTag.

Referenced by IScsiProcessLoginRsp().

```
EFI_STATUS IScsiCHAPToSendReq ( IN ISCSI_CONNECTION *   Conn,
                                IN OUT NET_BUF *        Pdu
                              )
```

This function fills the CHAP authentication information into the login PDU during the security negotiation stage in the iSCSI connection login.

Parameters:

[in]	*Conn*	The iSCSI connection.
[in,out]	*Pdu*	The PDU to send out.

Return values:

EFI_SUCCESS	All check passed and the phase-related CHAP authentication info is filled into the iSCSI PDU.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_PROTOCOL_ERROR	Some kind of protocol error occurred.

References _ISCSI_CHAP_AUTH_DATA::AuthConfig, _ISCSI_SESSION::AuthData, _ISCSI_SESSION::AuthType, _ISCSI_SESSION::CHAP, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPName, _ISCSI_CHAP_AUTH_DATA::CHAPResponse, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPType, _ISCSI_SESSION::ConfigData, ISCSI_PRIVATE_DATA::InitiatorName, ISCSI_AUTH_INITIAL, ISCSI_AUTH_METHOD_CHAP, ISCSI_AUTH_TYPE_NONE, ISCSI_CHAP_ALGORITHM_MD5, ISCSI_CHAP_MUTUAL, ISCSI_CHAP_RSP_LEN, ISCSI_CHAP_STEP_FOUR, ISCSI_CHAP_STEP_ONE, ISCSI_CHAP_STEP_THREE, ISCSI_CHAP_STEP_TWO, ISCSI_KEY_AUTH_METHOD, ISCSI_KEY_CHAP_ALGORITHM, ISCSI_KEY_CHAP_CHALLENGE, ISCSI_KEY_CHAP_IDENTIFIER, ISCSI_KEY_CHAP_NAME, ISCSI_KEY_CHAP_RESPONSE, ISCSI_KEY_INITIATOR_NAME, ISCSI_KEY_SESSION_TYPE, ISCSI_KEY_TARGET_NAME, ISCSI_KEY_VALUE_NONE, ISCSI_LOGIN_REQ_PDU_FLAG_TRANSIT, ISCSI_SECURITY_NEGOTIATION, ISCSI_SET_FLAG, IScsiAddKeyValuePair(), IScsiBinToHex(), IScsiGenRandom(), mPrivate, _ISCSI_CHAP_AUTH_DATA::OutChallenge, _ISCSI_CHAP_AUTH_DATA::OutChallengeLength, _ISCSI_CHAP_AUTH_DATA::OutIdentifier, _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData, and _ISCSI_SESSION_CONFIG_NVDATA::TargetName.

Referenced by IScsiPrepareLoginReq().

NetworkPkg/IScsiDxe/IScsiConfig.c File Reference

Functions

VOID IScsiIpToStr (IN EFI_IP_ADDRESS *Ip, IN BOOLEAN Ipv6Flag, OUT CHAR16 *Str)

BOOLEAN IpIsUnicast (IN EFI_IP_ADDRESS *Ip, IN UINT8 IpMode)

EFI_STATUS IScsiParseIsIdFromString (IN CONST CHAR16 *String, IN OUT UINT8 *IsId)

EFI_STATUS IScsiConvertIsIdToString (OUT CHAR16 *String, IN UINT8 *IsId)

ISCSI_ATTEMPT_CONFIG_NVDATA * IScsiConfigGetAttemptByConfigIndex (IN UINT8 AttemptConfigIndex)

ISCSI_ATTEMPT_CONFIG_NVDATA * IScsiConfigGetAttemptByNic (IN ISCSI_ATTEMPT_CONFIG_NVDATA *NewAttempt, IN UINT8 IScsiMode)

VOID IScsiConvertAttemptConfigDataToIfrNvData (IN ISCSI_ATTEMPT_CONFIG_NVDATA *Attempt, IN OUT ISCSI_CONFIG_IFR_NVDATA *IfrNvData)

EFI_STATUS IScsiConvertIfrNvDataToAttemptConfigData (IN ISCSI_CONFIG_IFR_NVDATA *IfrNvData, IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA *Attempt)

EFI_STATUS IScsiCreateOpCode (IN UINT16 StartLabelNumber, OUT VOID **StartOpCodeHandle, OUT EFI_IFR_GUID_LABEL **StartLabel, OUT VOID **EndOpCodeHandle, OUT EFI_IFR_GUID_LABEL **EndLabel)

EFI_STATUS IScsiConfigAddAttempt (VOID)

VOID IScsiConfigUpdateAttempt (VOID)

EFI_STATUS IScsiConfigDeleteAttempts (IN ISCSI_CONFIG_IFR_NVDATA *IfrNvData)

EFI_STATUS IScsiConfigDisplayDeleteAttempts (IN ISCSI_CONFIG_IFR_NVDATA *IfrNvData)

EFI_STATUS IScsiConfigDisplayOrderAttempts (VOID)

EFI_STATUS IScsiConfigOrderAttempts (IN ISCSI_CONFIG_IFR_NVDATA *IfrNvData)

EFI_STATUS IScsiConfigProcessDefault (IN EFI_QUESTION_ID KeyValue, IN ISCSI_CONFIG_IFR_NVDATA *IfrNvData)

EFI_STATUS EFIAPI IScsiFormExtractConfig (IN CONST EFI_HII_CONFIG_ACCESS_PROTOCOL *This, IN CONST EFI_STRING Request, OUT EFI_STRING *Progress, OUT EFI_STRING *Results)

EFI_STATUS EFIAPI IScsiFormRouteConfig (IN CONST EFI_HII_CONFIG_ACCESS_PROTOCOL *This, IN CONST EFI_STRING Configuration, OUT EFI_STRING *Progress)

EFI_STATUS EFIAPI IScsiFormCallback (IN CONST EFI_HII_CONFIG_ACCESS_PROTOCOL *This, IN EFI_BROWSER_ACTION Action, IN EFI_QUESTION_ID QuestionId, IN UINT8 Type, IN OUT EFI_IFR_TYPE_VALUE *Value, OUT EFI_BROWSER_ACTION_REQUEST *ActionRequest)

EFI_STATUS IScsiConfigFormInit (IN EFI_HANDLE DriverBindingHandle)

EFI_STATUS

IScsiConfigFormUnload (IN EFI_HANDLE DriverBindingHandle)

Variables

CHAR16 mVendorStorageName [] = L"ISCSI_CONFIG_IFR_NVDATA"
BOOLEAN mIScsiDeviceListUpdated = FALSE
UINTN mNumberOfIScsiDevices = 0
ISCSI_FORM_CALLBACK_INFO * mCallbackInfo = NULL
HII_VENDOR_DEVICE_PATH mIScsiHiiVendorDevicePath

Detailed Description

Helper functions for configuring or getting the parameters relating to iSCSI.

Copyright (c) 2004 - 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
BOOLEAN IpIsUnicast ( IN EFI_IP_ADDRESS *  Ip,
                      IN UINT8             IpMode
                    )
```

Check whether the input IP address is valid.

Parameters:

[in]	*Ip*	The IP address.
[in]	*IpMode*	Indicates iSCSI running on IP4 or IP6 stack.

Return values:

TRUE	The input IP address is valid.
FALSE	Otherwise

References IP_MODE_IP4, and IP_MODE_IP6.

Referenced by IScsiConvertIfrNvDataToAttemptConfigData(), and IScsiFormCallback().

```
EFI_STATUS IScsiConfigAddAttempt ( VOID )
```

Callback function when user presses "Add an Attempt".

Return values:

EFI_OUT_OF_RESOURCES	Does not have sufficient resources to finish this operation.
EFI_SUCCESS	The operation is completed successfully.

References ISCSI_NIC_INFO::BusNumber, ISCSI_NIC_INFO::DeviceNumber, FORMID_ATTEMPT_FORM, FORMID_MAC_FORM, ISCSI_NIC_INFO::FunctionNumber, gIScsiConfigGuid, ISCSI_NIC_INFO::HwAddressSize, ISCSI_MAX_MAC_STRING_LEN, ISCSI_NAME_IFR_MAX_SIZE, IScsiCreateOpCode(), IScsiMacAddrToStr(), KEY_MAC_ENTRY_BASE, MAC_ENTRY_LABEL, mPrivate, ISCSI_NIC_INFO::NicIndex, ISCSI_PRIVATE_DATA::NicInfoList, ISCSI_NIC_INFO::PermanentAddress, ISCSI_PRIVATE_DATA::PortString, _ISCSI_FORM_CALLBACK_INFO::RegisteredHandle, and ISCSI_NIC_INFO::VlanId.

Referenced by IScsiFormCallback().

EFI_STATUS IScsiConfigDeleteAttempts (IN ISCSI_CONFIG_IFR_NVDATA * *IfrNvData*)

Callback function when user presses "Commit Changes and Exit" in Delete Attempts.

Parameters:

`[in]` *IfrNvData* The IFR NV data.

Return values:

EFI_NOT_FOUND	Cannot find the corresponding variable.
EFI_SUCCESS	The operation is completed successfully.
EFI_ABOTRED	This operation is aborted cause of error configuration.
EFI_OUT_OF_RESOURCES	Fail to finish the operation due to lack of resources.

References _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptConfigIndex, ISCSI_PRIVATE_DATA::AttemptConfigs, ISCSI_PRIVATE_DATA::AttemptCount, _ISCSI_SESSION_CONFIG_NVDATA::Enabled, ISCSI_PRIVATE_DATA::EnableMpio, gIScsiConfigGuid, ISCSI_ENABLED, ISCSI_ENABLED_FOR_MPIO, ISCSI_MAX_MAC_STRING_LEN, IScsiGetVariableAndSize(), _ISCSI_ATTEMPT_CONFIG_NVDATA::Link, _ISCSI_ATTEMPT_CONFIG_NVDATA::MacString, ISCSI_PRIVATE_DATA::MpioCount, mPrivate, ISCSI_PRIVATE_DATA::PortString, _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData, and ISCSI_PRIVATE_DATA::SinglePathCount.

Referenced by IScsiFormCallback().

EFI_STATUS IScsiConfigDisplayDeleteAttempts (IN ISCSI_CONFIG_IFR_NVDATA * *IfrNvData*)

Callback function when user presses "Delete Attempts".

Parameters:

`[in]` *IfrNvData* The IFR nv data.

Return values:

EFI_INVALID_PARAMETER	Any parameter is invalid.
EFI_BUFFER_TOO_SMALL	The buffer in UpdateData is too small.
EFI_SUCCESS	The operation is completed successfully.

References ATTEMPT_DEL_QUESTION_ID, ATTEMPT_DEL_VAR_OFFSET, ISCSI_PRIVATE_DATA::AttemptConfigs, _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptTitleHelpToken, _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptTitleToken, CONFIGURATION_VARSTORE_ID, DELETE_ENTRY_LABEL, FORMID_DELETE_FORM, gIScsiConfigGuid, ISCSI_MAX_ATTEMPTS_NUM, IScsiCreateOpCode(), IScsiGetVariableAndSize(), mPrivate, and _ISCSI_FORM_CALLBACK_INFO::RegisteredHandle.

Referenced by IScsiFormCallback().

EFI_STATUS IScsiConfigDisplayOrderAttempts (VOID)

Callback function when user presses "Change Attempt Order".

Return values:

EFI_INVALID_PARAMETER	Any parameter is invalid.
EFI_OUT_OF_RESOURCES	Does not have sufficient resources to finish this operation.
EFI_SUCCESS	The operation is completed successfully.

References _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptConfigIndex, ISCSI_PRIVATE_DATA::AttemptConfigs, ISCSI_PRIVATE_DATA::AttemptCount, _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptTitleToken, CONFIGURATION_VARSTORE_ID, DYNAMIC_ORDERED_LIST_QUESTION_ID, DYNAMIC_ORDERED_LIST_VAR_OFFSET, FORMID_ORDER_FORM, gIScsiConfigGuid, ISCSI_MAX_ATTEMPTS_NUM, IScsiCreateOpCode(), mPrivate, ORDER_ENTRY_LABEL, and _ISCSI_FORM_CALLBACK_INFO::RegisteredHandle.

Referenced by IScsiFormCallback().

EFI_STATUS IScsiConfigFormInit (IN EFI_HANDLE *DriverBindingHandle*)

Initialize the iSCSI configuration form.

Parameters:

`[in]` *DriverBindingHandle*	The iSCSI driverbinding handle.

Return values:

EFI_SUCCESS	The iSCSI configuration form is initialized.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.

References _ISCSI_FORM_CALLBACK_INFO::ConfigAccess, _ISCSI_FORM_CALLBACK_INFO::Current, _ISCSI_FORM_CALLBACK_INFO::DriverHandle, gIScsiConfigGuid, ISCSI_FORM_CALLBACK_INFO_SIGNATURE, IScsiConfigVfrBin, IScsiDxeStrings, IScsiFormCallback(), IScsiFormExtractConfig(), IScsiFormRouteConfig(), _ISCSI_FORM_CALLBACK_INFO::RegisteredHandle, and _ISCSI_FORM_CALLBACK_INFO::Signature.

Referenced by IScsiDriverEntryPoint().

EFI_STATUS IScsiConfigFormUnload (IN EFI_HANDLE *DriverBindingHandle*)

Unload the iSCSI configuration form, this includes: delete all the iSCSI configuration entries, uninstall the form callback protocol, and free the resources used.

Parameters:

`[in]` *DriverBindingHandle*	The iSCSI driverbinding handle.

Return values:

EFI_SUCCESS	The iSCSI configuration form is unloaded.
Others	Failed to unload the form.

References ISCSI_PRIVATE_DATA::AttemptConfigs, ISCSI_PRIVATE_DATA::AttemptCount,

_ISCSI_FORM_CALLBACK_INFO::ConfigAccess, _ISCSI_FORM_CALLBACK_INFO::DriverHandle, mPrivate, ISCSI_PRIVATE_DATA::NewAttempt, ISCSI_PRIVATE_DATA::NicCount, ISCSI_PRIVATE_DATA::NicInfoList, and _ISCSI_FORM_CALLBACK_INFO::RegisteredHandle.

Referenced by IScsiDriverEntryPoint(), and IScsiUnload().

ISCSI_ATTEMPT_CONFIG_NVDATA* IScsiConfigGetAttemptByConfigIndex (IN UINT8 *AttemptConfigIndex*)

Get the attempt config data from global structure by the ConfigIndex.

Parameters:

`[in]` *AttemptConfigIndex* The unique index indicates the attempt.

Returns:

Pointer to the attempt config data.

Return values:

NULL The attempt configuration data cannot be found.

References _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptConfigIndex, ISCSI_PRIVATE_DATA::AttemptConfigs, and mPrivate.

Referenced by IScsiConfigOrderAttempts(), IScsiConfigProcessDefault(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiFillNICAndTargetSections(), and IScsiGetConfigData().

ISCSI_ATTEMPT_CONFIG_NVDATA* IScsiConfigGetAttemptByNic (IN ISCSI_ATTEMPT_CONFIG_NVDATA * *NewAttempt*, IN UINT8 *IScsiMode*)

Get the existing attempt config data from global structure by the NicIndex.

Parameters:

`[in]` *NewAttempt* The created new attempt

`[in]` *IScsiMode* The IScsi Mode of the new attempt, Enabled or Enabled for MPIO.

Returns:

Pointer to the existing attempt config data which has the same NICIndex as the new created attempt.

Return values:

NULL The attempt with NicIndex does not exist.

References ISCSI_PRIVATE_DATA::AttemptConfigs, _ISCSI_SESSION_CONFIG_NVDATA::Enabled, mPrivate, _ISCSI_ATTEMPT_CONFIG_NVDATA::NicIndex, and _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData.

Referenced by IScsiConvertIfrNvDataToAttemptConfigData().

EFI_STATUS IScsiConfigOrderAttempts (IN ISCSI_CONFIG_IFR_NVDATA * *IfrNvData*)

Callback function when user presses "Commit Changes and Exit" in Change Attempt Order.

Parameters:

[in] *IfrNvData* The IFR nv data.

Return values:

EFI_OUT_OF_RESOURCES	Does not have sufficient resources to finish this operation.
EFI_NOT_FOUND	Cannot find the corresponding variable.
EFI_SUCCESS	The operation is completed successfully.

References ISCSI_PRIVATE_DATA::AttemptConfigs, gIScsiConfigGuid, ISCSI_MAX_ATTEMPTS_NUM, IScsiConfigGetAttemptByConfigIndex(), IScsiGetVariableAndSize(), _ISCSI_ATTEMPT_CONFIG_NVDATA::Link, and mPrivate.

Referenced by IScsiFormCallback().

```
EFI_STATUS IScsiConfigProcessDefault ( IN EFI_QUESTION_ID              KeyValue,
                                       IN ISCSI_CONFIG_IFR_NVDATA *    IfrNvData
                                     )
```

Callback function when a user presses "Attempt *" or when a user selects a NIC to create the new attempt.

Parameters:

[in] *KeyValue* A unique value which is sent to the original exporting driver so that it can identify the type of data to expect.

[in] *IfrNvData* The IFR nv data.

Return values:

EFI_OUT_OF_RESOURCES	Does not have sufficient resources to finish this operation.
EFI_NOT_FOUND	Cannot find the corresponding variable.
EFI_SUCCESS	The operation is completed successfully.

References _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptConfigIndex, _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptName, _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptTitleHelpToken, _ISCSI_ATTEMPT_CONFIG_NVDATA::AuthConfigData, _ISCSI_ATTEMPT_CONFIG_NVDATA::AuthenticationType, ISCSI_NIC_INFO::BusNumber, _ISCSI_ATTEMPT_CONFIG_NVDATA::CHAP, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPType, CONNECT_DEFAULT_TIMEOUT, CONNECT_MIN_RETRY, _ISCSI_SESSION_CONFIG_NVDATA::ConnectRetryCount, _ISCSI_SESSION_CONFIG_NVDATA::ConnectTimeout, _ISCSI_FORM_CALLBACK_INFO::Current, ISCSI_NIC_INFO::DeviceNumber, ISCSI_NIC_INFO::FunctionNumber, gIScsiConfigGuid, ISCSI_NIC_INFO::HwAddressSize, ISCSI_AUTH_TYPE_CHAP, ISCSI_CHAP_UNI, ISCSI_MAX_ATTEMPTS_NUM, ISCSI_MAX_MAC_STRING_LEN, ISCSI_NAME_IFR_MAX_SIZE, ISCSI_WELL_KNOWN_PORT, IScsiConfigGetAttemptByConfigIndex(), IScsiConvertAttemptConfigDataToIfrNvData(), IScsiGetNicInfoByIndex(), IScsiGetVariableAndSize(), IScsiMacAddrToStr(), _ISCSI_SESSION_CONFIG_NVDATA::IsId, KEY_ATTEMPT_ENTRY_BASE, KEY_MAC_ENTRY_BASE, _ISCSI_ATTEMPT_CONFIG_NVDATA::MacString, ISCSI_PRIVATE_DATA::MaxNic, mPrivate, ISCSI_PRIVATE_DATA::NewAttempt, _ISCSI_ATTEMPT_CONFIG_NVDATA::NicIndex, ISCSI_NIC_INFO::PermanentAddress, ISCSI_PRIVATE_DATA::PortString, _ISCSI_FORM_CALLBACK_INFO::RegisteredHandle, _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData, _ISCSI_SESSION_CONFIG_NVDATA::TargetPort, and ISCSI_NIC_INFO::VlanId.

Referenced by IScsiFormCallback().

VOID IScsiConfigUpdateAttempt (VOID)

Update the MAIN form to display the configured attempts.

References ATTEMPT_ENTRY_LABEL, ATTEMPT_NAME_MAX_SIZE, _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptConfigIndex, ISCSI_PRIVATE_DATA::AttemptConfigs, _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptName, _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptTitleHelpToken, _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptTitleToken, FORMID_ATTEMPT_FORM, FORMID_MAIN_FORM, gIScsiConfigGuid, IScsiCreateOpCode(), KEY_ATTEMPT_ENTRY_BASE, mPrivate, ISCSI_PRIVATE_DATA::PortString, and _ISCSI_FORM_CALLBACK_INFO::RegisteredHandle.

Referenced by IScsiConvertIfrNvDataToAttemptConfigData(), IScsiFormCallback(), and IScsiGetConfigData().

VOID IScsiConvertAttemptConfigDataToIfrNvData (IN ISCSI_ATTEMPT_CONFIG_NVDATA * *Attempt*, IN OUT ISCSI_CONFIG_IFR_NVDATA * *IfrNvData*)

Convert the iSCSI configuration data into the IFR data.

Parameters:

[in]	*Attempt*	The iSCSI attempt config data.
[in,out]	*IfrNvData*	The IFR nv data.

References _ISCSI_SESSION_CONFIG_NVDATA::BootLun, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPName, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPSecret, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPType, _ISCSI_SESSION_CONFIG_NVDATA::ConnectRetryCount, _ISCSI_SESSION_CONFIG_NVDATA::ConnectTimeout, _ISCSI_SESSION_CONFIG_NVDATA::Enabled, _ISCSI_SESSION_CONFIG_NVDATA::Gateway, _ISCSI_SESSION_CONFIG_NVDATA::InitiatorInfoFromDhcp, IP_MODE_IP4, IP_MODE_IP6, _ISCSI_SESSION_CONFIG_NVDATA::IpMode, ISCSI_AUTH_TYPE_CHAP, IScsiConvertIsIdToString(), IScsiIpToStr(), IScsiLunToUnicodeStr(), _ISCSI_SESSION_CONFIG_NVDATA::IsId, _ISCSI_SESSION_CONFIG_NVDATA::LocalIp, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::ReverseCHAPName, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::ReverseCHAPSecret, _ISCSI_SESSION_CONFIG_NVDATA::SubnetMask, _ISCSI_SESSION_CONFIG_NVDATA::TargetInfoFromDhcp, _ISCSI_SESSION_CONFIG_NVDATA::TargetIp, _ISCSI_SESSION_CONFIG_NVDATA::TargetName, and _ISCSI_SESSION_CONFIG_NVDATA::TargetPort.

Referenced by IScsiConfigProcessDefault(), and IScsiFormExtractConfig().

EFI_STATUS IScsiConvertIfrNvDataToAttemptConfigData (IN ISCSI_CONFIG_IFR_NVDATA * *IfrNvData*, IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA * *Attempt*)

Convert the IFR data to iSCSI configuration data.

Parameters:

[in]	*IfrNvData*	Point to ISCSI_CONFIG_IFR_NVDATA.
[in,out]	*Attempt*	The iSCSI attempt config data.

Return values:

EFI_INVALID_PARAMETER	Any input or configured parameter is invalid.
EFI_NOT_FOUND	Cannot find the corresponding variable.
EFI_OUT_OF_RESOURCES	The operation is failed due to lack of resources.
EFI_ABORTED	The operation is aborted.
EFI_SUCCESS	The operation is completed successfully.

References ATTEMPT_NAME_MAX_SIZE, ATTEMPT_NAME_SIZE, ISCSI_PRIVATE_DATA::AttemptConfigs, ISCSI_PRIVATE_DATA::AttemptCount, _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptName, ISCSI_NIC_INFO::BusNumber, CONNECT_MIN_TIMEOUT, _ISCSI_CONFIG_IFR_NVDATA::ConnectRetryCount, ISCSI_NIC_INFO::DeviceNumber, ISCSI_PRIVATE_DATA::EnableMpio, ISCSI_NIC_INFO::FunctionNumber, gIScsiConfigGuid, IP_MODE_AUTOCONFIG, IP_MODE_IP4, IP_MODE_IP6, IpIsUnicast(), IpMode, ISCSI_AUTH_TYPE_CHAP, ISCSI_CHAP_MUTUAL, ISCSI_CONFIG_VAR_ATTR, ISCSI_DISABLED, ISCSI_ENABLED, ISCSI_ENABLED_FOR_MPIO, ISCSI_MAX_MAC_STRING_LEN, ISCSI_NAME_IFR_MAX_SIZE, ISCSI_WELL_KNOWN_PORT, IScsiConfigGetAttemptByConfigIndex(), IScsiConfigGetAttemptByNic(), IScsiConfigUpdateAttempt(), IScsiGetNicInfoByIndex(), IScsiGetVariableAndSize(), ISCSI_PRIVATE_DATA::MpioCount, mPrivate, ISCSI_PRIVATE_DATA::NewAttempt, ISCSI_PRIVATE_DATA::PortString, _ISCSI_FORM_CALLBACK_INFO::RegisteredHandle, and ISCSI_PRIVATE_DATA::SinglePathCount.

Referenced by IScsiFormCallback().

```
EFI_STATUS IScsiConvertIsIdToString ( OUT CHAR16 *   String,
                                      IN UINT8 *     IsId
                                    )
```

Convert IsId from binary to string format.

Parameters:

[out]	*String*	The buffer to store the converted string.
[in]	*IsId*	The buffer to store IsId.

Return values:

EFI_SUCCESS	The string converted successfully.
EFI_INVALID_PARAMETER	Any input parameter is invalid.

References ISID_CONFIGURABLE_STORAGE.

Referenced by IScsiConvertAttemptConfigDataToIfrNvData(), and IScsiFormCallback().

```
EFI_STATUS IScsiCreateOpCode ( IN UINT16                    StartLabelNumber,
                               OUT VOID **                  StartOpCodeHandle,
                               OUT EFI_IFR_GUID_LABEL **    StartLabel,
                               OUT VOID **                  EndOpCodeHandle,
```

OUT EFI_IFR_GUID_LABEL ** *EndLabel*
)

Create Hii Extend Label OpCode as the start opcode and end opcode. It is a help function.

Parameters:

[in]	*StartLabelNumber*	The number of start label.
[out]	*StartOpCodeHandle*	Points to the start opcode handle.
[out]	*StartLabel*	Points to the created start opcode.
[out]	*EndOpCodeHandle*	Points to the end opcode handle.
[out]	*EndLabel*	Points to the created end opcode.

Return values:

EFI_OUT_OF_RESOURCES	Do not have sufficient resource to finish this operation.
EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_SUCCESS	The operation is completed successfully.

References LABEL_END.

Referenced by IScsiConfigAddAttempt(), IScsiConfigDisplayDeleteAttempts(), IScsiConfigDisplayOrderAttempts(), and IScsiConfigUpdateAttempt().

EFI_STATUS EFIAPI IScsiFormCallback (
IN CONST EFI_HII_CONFIG_ACCESS_PROTOCOL * *This*,
IN EFI_BROWSER_ACTION *Action*,
IN EFI_QUESTION_ID *QuestionId*,
IN UINT8 *Type*,
IN OUT EFI_IFR_TYPE_VALUE * *Value*,
OUT EFI_BROWSER_ACTION_REQUEST * *ActionRequest*
)

This function is called to provide results data to the driver. This data consists of a unique key that is used to identify which data is either being passed back or being asked for.

Parameters:

[in]	*This*	Points to the EFI_HII_CONFIG_ACCESS_PROTOCOL.
[in]	*Action*	Specifies the type of action taken by the browser.
[in]	*QuestionId*	A unique value which is sent to the original exporting driver so that it can identify the type of data to expect. The format of the data tends to vary based on the opcode that generated the callback.
[in]	*Type*	The type of value for the question.
[in,out]	*Value*	A pointer to the data being sent to the original exporting driver.
[out]	*ActionRequest*	On return, points to the action requested by the callback function.

Return values:

EFI_SUCCESS	The callback successfully handled the action.
EFI_OUT_OF_RESOURCES	Not enough storage is available to hold the variable and its data.
EFI_DEVICE_ERROR	The variable could not be saved.
EFI_UNSUPPORTED	The specified Action is not supported by the callback.

References ATTEMPT_NAME_SIZE, _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptName, _ISCSI_ATTEMPT_CONFIG_NVDATA::AuthConfigData,

_ISCSI_ATTEMPT_CONFIG_NVDATA::AutoConfigureMode, _ISCSI_SESSION_CONFIG_NVDATA::BootLun, _ISCSI_ATTEMPT_CONFIG_NVDATA::CHAP, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPName, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPSecret, _ISCSI_FORM_CALLBACK_INFO::Current, _ISCSI_CONFIG_IFR_NVDATA::DeleteAttemptList, _ISCSI_CONFIG_IFR_NVDATA::DynamicOrderedList, _ISCSI_SESSION_CONFIG_NVDATA::Gateway, gIScsiInitiatorName, ISCSI_PRIVATE_DATA::InitiatorName, ISCSI_PRIVATE_DATA::InitiatorNameLength, IP_MODE_IP4, IP_MODE_IP6, IP_STR_MAX_SIZE, IpIsUnicast(), ISCSI_AUTH_TYPE_CHAP, ISCSI_CHAP_UNI, ISCSI_FORM_CALLBACK_INFO_FROM_FORM_CALLBACK, ISCSI_LUN_STR_MAX_LEN, ISCSI_NAME_MAX_SIZE, IScsiAsciiStrToIp(), IScsiAsciiStrToLun(), IScsiConfigAddAttempt(), IScsiConfigDeleteAttempts(), IScsiConfigDisplayDeleteAttempts(), IScsiConfigDisplayOrderAttempts(), IScsiConfigOrderAttempts(), IScsiConfigProcessDefault(), IScsiConfigUpdateAttempt(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiConvertIsIdToString(), IScsiGetSubnetMaskPrefixLength(), IScsiIpToStr(), IScsiNormalizeName(), IScsiParseIsIdFromString(), _ISCSI_SESSION_CONFIG_NVDATA::IsId, KEY_ADD_ATTEMPT, KEY_ATTEMPT_NAME, KEY_AUTH_TYPE, KEY_BOOT_LUN, KEY_CHAP_NAME, KEY_CHAP_SECRET, KEY_CONFIG_ISID, KEY_DELETE_ATTEMPT, KEY_DHCP_ENABLE, KEY_GATE_WAY, KEY_IGNORE_DELETE_ATTEMPT, KEY_IGNORE_ORDER_CHANGES, KEY_INITIATOR_NAME, KEY_IP_MODE, KEY_LOCAL_IP, KEY_ORDER_ATTEMPT_CONFIG, KEY_REVERSE_CHAP_NAME, KEY_REVERSE_CHAP_SECRET, KEY_SAVE_ATTEMPT_CONFIG, KEY_SAVE_DELETE_ATTEMPT, KEY_SAVE_ORDER_CHANGES, KEY_SUBNET_MASK, KEY_TARGET_IP, KEY_TARGET_NAME, _ISCSI_SESSION_CONFIG_NVDATA::LocalIp, mPrivate, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::ReverseCHAPName, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::ReverseCHAPSecret, _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData, _ISCSI_SESSION_CONFIG_NVDATA::SubnetMask, _ISCSI_SESSION_CONFIG_NVDATA::TargetIp, and _ISCSI_SESSION_CONFIG_NVDATA::TargetName.

Referenced by IScsiConfigFormInit().

```
EFI_STATUS EFIAPI
IScsiFormExtractConfig (
  IN CONST EFI_HII_CONFIG_ACCESS_PROTOCOL *  This,
  IN CONST EFI_STRING                        Request,
  OUT EFI_STRING *                           Progress,
  OUT EFI_STRING *                           Results
)
```

This function allows the caller to request the current configuration for one or more named elements. The resulting string is in <ConfigAltResp> format. Also, any and all alternative configuration strings shall be appended to the end of the current configuration string. If they are, they must appear after the current configuration. They must contain the same routing (GUID, NAME, PATH) as the current configuration string. They must have an additional description indicating the type of alternative configuration the string represents, "ALTCFG=<StringToken>". That <StringToken> (when converted from Hex UNICODE to binary) is a reference to a string in the associated string pack.

Parameters:

[in]	*This*	Points to the EFI_HII_CONFIG_ACCESS_PROTOCOL.
[in]	*Request*	A null-terminated Unicode string in <ConfigRequest> format. Note that this includes the routing information as well as the configurable name / value pairs. It is invalid for this string to be in <MultiConfigRequest> format.
[out]	*Progress*	On return, points to a character in the Request string. Points to the string's null terminator if request was successful. Points to the most recent "&" before the first

failing name / value pair (or the beginning of the string if the failure is in the first name / value pair) if the request was not successful.

`[out]` *Results* A null-terminated Unicode string in <ConfigAltResp> format which has all values filled in for the names in the Request string. String to be allocated by the called function.

Return values:

EFI_SUCCESS	The Results string is filled with the values corresponding to all requested names.
EFI_OUT_OF_RESOURCES	Not enough memory to store the parts of the results that must be stored awaiting possible future protocols.
EFI_INVALID_PARAMETER	For example, passing in a NULL for the Request parameter would result in this type of error. In this case, the Progress parameter would be set to NULL.
EFI_NOT_FOUND	Routing data doesn't match any known driver. Progress set to the first character in the routing header. Note: There is no requirement that the driver validate the routing data. It must skip the <ConfigHdr> in order to process the names.
EFI_INVALID_PARAMETER	Illegal syntax. Progress set to most recent "&" before the error or the beginning of the string.
EFI_INVALID_PARAMETER	Unknown name. Progress points to the & before the name in question.

References _ISCSI_FORM_CALLBACK_INFO::Current, _ISCSI_FORM_CALLBACK_INFO::DriverHandle, gIScsiConfigGuid, gIScsiInitiatorName, _ISCSI_CONFIG_IFR_NVDATA::InitiatorName, ISCSI_FORM_CALLBACK_INFO_FROM_FORM_CALLBACK, ISCSI_NAME_MAX_SIZE, IScsiConvertAttemptConfigDataToIfrNvData(), and mVendorStorageName.

Referenced by IScsiConfigFormInit().

```
EFI_STATUS EFIAPI
IScsiFormRouteConfig (
  IN CONST EFI_HII_CONFIG_ACCESS_PROTOCOL *  This,
  IN CONST EFI_STRING                        Configuration,
  OUT EFI_STRING *                           Progress
)
```

This function applies changes in a driver's configuration. Input is a Configuration, which has the routing data for this driver followed by name / value configuration pairs. The driver must apply those pairs to its configurable storage. If the driver's configuration is stored in a linear block of data and the driver's name / value pairs are in <BlockConfig> format, it may use the ConfigToBlock helper function (above) to simplify the job.

Parameters:

`[in]` *This* Points to the EFI_HII_CONFIG_ACCESS_PROTOCOL.

`[in]` *Configuration* A null-terminated Unicode string in <ConfigString> format.

`[out]` *Progress* A pointer to a string filled in with the offset of the most recent '&' before the first failing name / value pair (or the beginning of the string if the failure is in the first name / value pair) or the terminating NULL if all was successful.

Return values:

EFI_SUCCESS	The results have been distributed or are awaiting distribution.
EFI_OUT_OF_RESOURCES	Not enough memory to store the parts of the results that must be stored awaiting possible future protocols.

EFI_INVALID_PARAMETERS	Passing in a NULL for the Results parameter would result in this type of error.
EFI_NOT_FOUND	Target for the specified routing data was not found.

References gIScsiConfigGuid, and mVendorStorageName.

Referenced by IScsiConfigFormInit().

```
VOID IScsiIpToStr ( IN EFI_IP_ADDRESS *  Ip,
                    IN BOOLEAN           Ipv6Flag,
                    OUT CHAR16 *         Str
                  )
```

Convert the IP address into a dotted string.

Parameters:

[in]	*Ip*	The IP address.
[in]	*Ipv6Flag*	Indicates whether the IP address is version 4 or version 6.
[out]	*Str*	The formatted IP string.

References IP4_STR_MAX_SIZE, and IP_STR_MAX_SIZE.

Referenced by IScsiConvertAttemptConfigDataToIfrNvData(), and IScsiFormCallback().

```
EFI_STATUS IScsiParseIsIdFromString ( IN CONST CHAR16 *  String,
                                      IN OUT UINT8 *     IsId
                                    )
```

Parse IsId in string format and convert it to binary.

Parameters:

[in]	*String*	The buffer of the string to be parsed.
[in,out]	*IsId*	The buffer to store IsId.

Return values:

EFI_SUCCESS	The operation finished successfully.
EFI_INVALID_PARAMETER	Any input parameter is invalid.

References ISCSI_NAME_IFR_MAX_SIZE.

Referenced by IScsiFormCallback().

Variable Documentation

ISCSI_FORM_CALLBACK_INFO* mCallbackInfo = NULL

Referenced by IScsiGetConfigData().

BOOLEAN mIScsiDeviceListUpdated = FALSE

HII_VENDOR_DEVICE_PATH mIScsiHiiVendorDevicePath

Initial value:

```
{
  {
    {
      HARDWARE_DEVICE_PATH,
      HW_VENDOR_DP,
      {
        (UINT8) (sizeof (VENDOR_DEVICE_PATH)),
        (UINT8) ((sizeof (VENDOR_DEVICE_PATH)) >> 8)
      }
    },
    ISCSI_CONFIG_GUID
  },
  {
    END_DEVICE_PATH_TYPE,
    END_ENTIRE_DEVICE_PATH_SUBTYPE,
    {
      (UINT8) (END_DEVICE_PATH_LENGTH),
      (UINT8) ((END_DEVICE_PATH_LENGTH) >> 8)
    }
  }
}
```

UINTN mNumberOfIScsiDevices = 0

CHAR16 mVendorStorageName[] = L"ISCSI_CONFIG_IFR_NVDATA"

Referenced by IScsiFormExtractConfig(), and IScsiFormRouteConfig().

NetworkPkg/IScsiDxe/IScsiConfig.h File Reference

Data Structures

Defines

"memItemRight" valign="bottom">VAR_OFFSET(Field) ((UINT16) ((UINTN) &(((ISCSI_CONFIG_IFR_NVDA

Typedefs

_ISCSI_FORM_C

Functions

ISCSI_ATTEMPT_CON

Variables

ISCSI_FORM_CA

Detailed Description

The header file of functions for configuring or getting the parameters relating to iSCSI.

Copyright (c) 2004 - 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by IScsiConfigDisplayDeleteAttempts().

#define ATTEMPT_DEL_VAR_OFFSET VAR_OFFSET (DeleteAttemptList)

Referenced by IScsiConfigDisplayDeleteAttempts().

#define DYNAMIC_ONE_OF_VAR_OFFSET VAR_OFFSET (Enabled)

#define DYNAMIC_ORDERED_LIST_QUESTION_ID QUESTION_ID (DynamicOrderedList)

Referenced by IScsiConfigDisplayOrderAttempts().

#define DYNAMIC_ORDERED_LIST_VAR_OFFSET VAR_OFFSET (DynamicOrderedList)

Referenced by IScsiConfigDisplayOrderAttempts().

#define ISCSI_CONFIG_VAR_ATTR (EFI_VARIABLE_BOOTSERVICE_ACCESS | EFI_VARIABLE_NON_VOLATILE)

Referenced by IScsiConvertIfrNvDataToAttemptConfigData(), IScsiGetConfigData(), and IScsiStart().

#define ISCSI_FORM_CALLBACK_INFO_FROM_FORM_CALLBACK (Callback)

Value:

```
CR ( \
  Callback, \
  ISCSI_FORM_CALLBACK_INFO, \
  ConfigAccess, \
  ISCSI_FORM_CALLBACK_INFO_SIGNATURE \
  )
```

Referenced by IScsiFormCallback(), and IScsiFormExtractConfig().

#define ISCSI_FORM_CALLBACK_INFO_SIGNATURE SIGNATURE_32 ('I', 'f', 'c', 'i')

Referenced by IScsiConfigFormInit().

#define ISCSI_INITATOR_NAME_VAR_NAME L"I_NAME"

#define ISCSI_MAX_MAC_STRING_LEN 96

Referenced by IScsiConfigAddAttempt(), IScsiConfigDeleteAttempts(), IScsiConfigProcessDefault(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiDhcpIsConfigured(), IScsiGetConfigData(), and IScsiStart().

#define QUESTION_ID (Field) ((UINT16) (VAR_OFFSET (Field) + CONFIG_OPTION_OFFSET))

#define VAR_OFFSET (Field) ((UINT16) ((UINTN) &(((ISCSI_CONFIG_IFR_NVDATA *) 0)->Field)))

Typedef Documentation

typedef struct _ISCSI_FORM_CALLBACK_INFO ISCSI_FORM_CALLBACK_INFO

Function Documentation

EFI_STATUS IScsiConfigFormInit (IN EFI_HANDLE *DriverBindingHandle*)
Initialize the iSCSI configuration form.

Parameters:

`[in]` *DriverBindingHandle* The iSCSI driverbinding handle.

Return values:

EFI_SUCCESS	The iSCSI configuration form is initialized.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.

References _ISCSI_FORM_CALLBACK_INFO::ConfigAccess, _ISCSI_FORM_CALLBACK_INFO::Current, _ISCSI_FORM_CALLBACK_INFO::DriverHandle, gIScsiConfigGuid, ISCSI_FORM_CALLBACK_INFO_SIGNATURE, IScsiConfigVfrBin, IScsiDxeStrings, IScsiFormCallback(), IScsiFormExtractConfig(), IScsiFormRouteConfig(), _ISCSI_FORM_CALLBACK_INFO::RegisteredHandle, and _ISCSI_FORM_CALLBACK_INFO::Signature.

Referenced by IScsiDriverEntryPoint().

EFI_STATUS IScsiConfigFormUnload (IN EFI_HANDLE *DriverBindingHandle*)

Unload the iSCSI configuration form, this includes: delete all the iSCSI configuration entries, uninstall the form callback protocol, and free the resources used.

Parameters:

`[in]` *DriverBindingHandle* The iSCSI driverbinding handle.

Return values:

EFI_SUCCESS The iSCSI configuration form is unloaded.
Others Failed to unload the form.

References ISCSI_PRIVATE_DATA::AttemptConfigs, ISCSI_PRIVATE_DATA::AttemptCount, _ISCSI_FORM_CALLBACK_INFO::ConfigAccess, _ISCSI_FORM_CALLBACK_INFO::DriverHandle, mPrivate, ISCSI_PRIVATE_DATA::NewAttempt, ISCSI_PRIVATE_DATA::NicCount, ISCSI_PRIVATE_DATA::NicInfoList, and _ISCSI_FORM_CALLBACK_INFO::RegisteredHandle.

Referenced by IScsiDriverEntryPoint(), and IScsiUnload().

ISCSI_ATTEMPT_CONFIG_NVDATA* IScsiConfigGetAttemptByConfigIndex (IN UINT8 *AttemptConfigIndex*)

Get the attempt config data from global structure by the ConfigIndex.

Parameters:

`[in]` *AttemptConfigIndex* The unique index indicates the attempt.

Returns:

Pointer to the attempt config data.

Return values:

NULL The attempt configuration data can not be found.

Get the attempt config data from global structure by the ConfigIndex.

Parameters:

`[in]` *AttemptConfigIndex* The unique index indicates the attempt.

Returns:

Pointer to the attempt config data.

Return values:

NULL The attempt configuration data cannot be found.

References _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptConfigIndex, ISCSI_PRIVATE_DATA::AttemptConfigs, and mPrivate.

Referenced by IScsiConfigOrderAttempts(), IScsiConfigProcessDefault(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiFillNICAndTargetSections(), and IScsiGetConfigData().

VOID IScsiConfigUpdateAttempt (VOID)

Update the MAIN form to display the configured attempts.

References ATTEMPT_ENTRY_LABEL, ATTEMPT_NAME_MAX_SIZE, _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptConfigIndex, ISCSI_PRIVATE_DATA::AttemptConfigs,

_ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptName, _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptTitleHelpToken, _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptTitleToken, FORMID_ATTEMPT_FORM, FORMID_MAIN_FORM, gIScsiConfigGuid, IScsiCreateOpCode(), KEY_ATTEMPT_ENTRY_BASE, mPrivate, ISCSI_PRIVATE_DATA::PortString, and _ISCSI_FORM_CALLBACK_INFO::RegisteredHandle.

Referenced by IScsiConvertIfrNvDataToAttemptConfigData(), IScsiFormCallback(), and IScsiGetConfigData().

Variable Documentation

UINT8 IScsiConfigVfrBin[]

Referenced by IScsiConfigFormInit().

UINT8 IScsiDxeStrings[]

Referenced by IScsiConfigFormInit().

ISCSI_FORM_CALLBACK_INFO* mCallbackInfo

Referenced by IScsiGetConfigData().

NetworkPkg/IScsiDxe/IScsiConfigNVDataStruc.h File Reference

Data Structures

struct _ISCSI_CONFIG_IFR_NVDATA

Defines

"memItemRight" valign="bottom">VAR_EQ_TEST_NAME 0x100

#define CONFIGURATION_VARSTORE_ID 0x6666
#define FORMID_MAIN_FORM 1
#define FORMID_MAC_FORM 2
#define FORMID_ATTEMPT_FORM 3
#define FORMID_ORDER_FORM 4
#define FORMID_DELETE_FORM 5
#define ISCSI_NAME_IFR_MIN_SIZE 4
#define ISCSI_NAME_IFR_MAX_SIZE 223
#define ISCSI_NAME_MAX_SIZE 224
#define ATTEMPT_NAME_MAX_SIZE 96
#define ATTEMPT_NAME_SIZE 10
#define CONNECT_MIN_RETRY 0
#define CONNECT_MAX_RETRY 16
#define CONNECT_MIN_TIMEOUT 100
#define CONNECT_MAX_TIMEOUT 20000
#define CONNECT_DEFAULT_TIMEOUT 1000
#define ISCSI_MAX_ATTEMPTS_NUM 255
#define ISCSI_DISABLED 0
#define ISCSI_ENABLED 1
#define ISCSI_ENABLED_FOR_MPIO 2
#define IP_MODE_IP4 0
#define IP_MODE_IP6 1
#define IP_MODE_AUTOCONFIG 2
#define ISCSI_AUTH_TYPE_NONE 0
#define ISCSI_AUTH_TYPE_CHAP 1
#define ISCSI_AUTH_TYPE_KRB 2
#define IP4_MIN_SIZE 7
#define IP4_MAX_SIZE 15
#define IP4_STR_MAX_SIZE 16
#define IP_MIN_SIZE 2
#define IP_MAX_SIZE 39
#define IP_STR_MAX_SIZE 40
#define LUN_MIN_SIZE 1
#define LUN_MAX_SIZE 20
#define ISCSI_CHAP_UNI 0
#define ISCSI_CHAP_MUTUAL 1
#define TARGET_PORT_MIN_NUM 0
#define TARGET_PORT_MAX_NUM 65535
#define LABEL_END 0xffff
#define KEY_INITIATOR_NAME 0x101
#define KEY_DHCP_ENABLE 0x102

#define KEY_LOCAL_IP 0x103
#define KEY_SUBNET_MASK 0x104
#define KEY_GATE_WAY 0x105
#define KEY_TARGET_IP 0x106
#define KEY_CHAP_NAME 0x107
#define KEY_CHAP_SECRET 0x108
#define KEY_REVERSE_CHAP_NAME 0x109
#define KEY_REVERSE_CHAP_SECRET 0x10a
#define KEY_SAVE_CHANGES 0x10b
#define KEY_TARGET_NAME 0x10c
#define KEY_BOOT_LUN 0x10d
#define KEY_ADD_ATTEMPT 0x10e
#define KEY_SAVE_ATTEMPT_CONFIG 0x10f
#define KEY_ORDER_ATTEMPT_CONFIG 0x110
#define KEY_SAVE_ORDER_CHANGES 0x111
#define KEY_IGNORE_ORDER_CHANGES 0x112
#define KEY_ATTEMPT_NAME 0x113
#define KEY_SAVE_DELETE_ATTEMPT 0x114
#define KEY_IGNORE_DELETE_ATTEMPT 0x115
#define KEY_DELETE_ATTEMPT 0x116
#define KEY_IP_MODE 0x11c
#define KEY_AUTH_TYPE 0x11d
#define KEY_CONFIG_ISID 0x11e
#define ATTEMPT_ENTRY_LABEL 0x9000
#define KEY_ATTEMPT_ENTRY_BASE 0xa000
#define KEY_DE_ATTEMPT_ENTRY_BASE 0xb000
#define KEY_DEVICE_ENTRY_BASE 0x1000
#define KEY_MAC_ENTRY_BASE 0x2000
#define MAC_ENTRY_LABEL 0x3000
#define ORDER_ENTRY_LABEL 0x4000
#define DELETE_ENTRY_LABEL 0x5000
#define CONFIG_OPTION_OFFSET 0x9000
#define ISCSI_LUN_STR_MAX_LEN 21
#define ISCSI_CHAP_SECRET_MIN_LEN 12
#define ISCSI_CHAP_SECRET_MAX_LEN 16
#define ISCSI_CHAP_SECRET_STORAGE 17
#define ISCSI_CHAP_NAME_MAX_LEN 126
#define ISCSI_CHAP_NAME_STORAGE 127
#define KERBEROS_SECRET_MIN_LEN 12
#define KERBEROS_SECRET_MAX_LEN 16
#define KERBEROS_SECRET_STORAGE 17
#define KERBEROS_NAME_MAX_LEN 96
#define KERBEROS_KDC_PORT_MIN_NUM 0
#define KERBEROS_KDC_PORT_MAX_NUM 65535
#define ISID_CONFIGURABLE_MIN_LEN 6
#define ISID_CONFIGURABLE_MAX_LEN 12
#define ISID_CONFIGURABLE_STORAGE 13

Typedefs

typedef struct
_ISCSI_CONFIG_IFR_NVDATA ISCSI_CONFIG_IFR_NVDATA

Detailed Description

Define NVData structures used by the iSCSI configuration component.

Copyright (c) 2004 - 2013, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"el"
href="../../d1/d98/temp0451.html#ca820eeca5ffaf1e7469db3754da3a5b">ATTEMPT_ENTRY_LABEL 0x9000

Referenced by IScsiConfigUpdateAttempt().

#define ATTEMPT_NAME_MAX_SIZE 96

Referenced by IScsiConfigUpdateAttempt(), and IScsiConvertIfrNvDataToAttemptConfigData().

#define ATTEMPT_NAME_SIZE 10

Referenced by IScsiConvertIfrNvDataToAttemptConfigData(), and IScsiFormCallback().

#define CONFIG_OPTION_OFFSET 0x9000

#define CONFIGURATION_VARSTORE_ID 0x6666

Referenced by IScsiConfigDisplayDeleteAttempts(), and IScsiConfigDisplayOrderAttempts().

#define CONNECT_DEFAULT_TIMEOUT 1000

Referenced by IScsiConfigProcessDefault().

#define CONNECT_MAX_RETRY 16

#define CONNECT_MAX_TIMEOUT 20000

#define CONNECT_MIN_RETRY 0

Referenced by IScsiConfigProcessDefault().

#define CONNECT_MIN_TIMEOUT 100

Referenced by IScsiConvertIfrNvDataToAttemptConfigData().

#define DELETE_ENTRY_LABEL 0x5000

Referenced by IScsiConfigDisplayDeleteAttempts().

#define FORMID_ATTEMPT_FORM 3

Referenced by IScsiConfigAddAttempt(), and IScsiConfigUpdateAttempt().

#define FORMID_DELETE_FORM 5

Referenced by IScsiConfigDisplayDeleteAttempts().

#define FORMID_MAC_FORM 2

Referenced by IScsiConfigAddAttempt().

#define FORMID_MAIN_FORM 1

#define FORMID_ORDER_FORM 4

Referenced by IScsiConfigDisplayOrderAttempts().

#define IP4_MAX_SIZE 15

#define IP4_MIN_SIZE 7

#define IP4_STR_MAX_SIZE 16

Referenced by IScsiIpToStr().

#define IP_MAX_SIZE 39

#define IP_MIN_SIZE 2

#define IP_MODE_AUTOCONFIG 2

Referenced by IScsiAsciiStrToIp(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiDhcp6ExtractRootPath(), IScsiDhcpExtractRootPath(), IScsiDhcpIsConfigured(), IScsiFillNICAndTargetSections(), IScsiGetConfigData(), IScsiStart(), and IScsiUpdateTargetAddress().

#define IP_MODE_IP4 0

Referenced by IpIsUnicast(), IScsiAsciiStrToIp(), IScsiConvertAttemptConfigDataToIfrNvData(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiDhcpIsConfigured(), IScsiFillNICAndTargetSections(), IScsiFormCallback(), IScsiGetConfigData(), and IScsiStart().

#define IP_MODE_IP6 1

Referenced by IpIsUnicast(), IScsiAsciiStrToIp(), IScsiConvertAttemptConfigDataToIfrNvData(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiCreateConnection(), IScsiDhcpIsConfigured(), IScsiFillNICAndTargetSections(), IScsiFormCallback(), IScsiGetConfigData(), and IScsiStart().

#define IP_STR_MAX_SIZE 40

Referenced by IScsiFormCallback(), and IScsiIpToStr().

#define ISCSI_AUTH_TYPE_CHAP 1

Referenced by IScsiCHAPOnRspReceived(), IScsiConfigProcessDefault(), IScsiConvertAttemptConfigDataToIfrNvData(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiExtScsiPassThruBuildDevicePath(), IScsiFillNICAndTargetSections(), IScsiFormCallback(), and IScsiStart().

#define ISCSI_AUTH_TYPE_KRB 2

Referenced by IScsiFillNICAndTargetSections(), IScsiGetConfigData(), IScsiPrepareLoginReq(), IScsiProcessLoginRsp(), IScsiRemoveNic(), and IScsiStart().

#define ISCSI_AUTH_TYPE_NONE 0

Referenced by IScsiCHAPOnRspReceived(), IScsiCHAPToSendReq(), IScsiExtScsiPassThruBuildDevicePath(), and IScsiFillNICAndTargetSections().

#define ISCSI_CHAP_MUTUAL 1

Referenced by IScsiCHAPOnRspReceived(), IScsiCHAPToSendReq(), IScsiConvertIfrNvDataToAttemptConfigData(), and IScsiFillNICAndTargetSections().

#define ISCSI_CHAP_NAME_MAX_LEN 126

#define ISCSI_CHAP_NAME_STORAGE 127

#define ISCSI_CHAP_SECRET_MAX_LEN 16

#define ISCSI_CHAP_SECRET_MIN_LEN 12

Referenced by IScsiCHAPCalculateResponse().

#define ISCSI_CHAP_SECRET_STORAGE 17

#define ISCSI_CHAP_UNI 0

Referenced by IScsiConfigProcessDefault(), IScsiExtScsiPassThruBuildDevicePath(), IScsiFillNICAndTargetSections(), and IScsiFormCallback().

#define ISCSI_DISABLED 0

Referenced by IScsiConvertIfrNvDataToAttemptConfigData(), IScsiDhcpIsConfigured(), IScsiFillNICAndTargetSections(), IScsiGetConfigData(), and IScsiStart().

#define ISCSI_ENABLED 1

Referenced by IScsiConfigDeleteAttempts(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiGetConfigData(), IScsiRemoveNic(), and IScsiStart().

#define ISCSI_ENABLED_FOR_MPIO 2

Referenced by IScsiConfigDeleteAttempts(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiFillNICAndTargetSections(), IScsiGetConfigData(), IScsiRemoveNic(), and IScsiStart().

#define ISCSI_LUN_STR_MAX_LEN 21

Referenced by IScsiFormCallback().

#define ISCSI_MAX_ATTEMPTS_NUM 255

Referenced by IScsiConfigDisplayDeleteAttempts(), IScsiConfigDisplayOrderAttempts(), IScsiConfigOrderAttempts(), and IScsiConfigProcessDefault().

#define ISCSI_NAME_IFR_MAX_SIZE 223

Referenced by IScsiConfigAddAttempt(), IScsiConfigProcessDefault(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiDhcpIsConfigured(), IScsiGetConfigData(), IScsiParseIsIdFromString(), and IScsiStart().

#define ISCSI_NAME_IFR_MIN_SIZE 4

#define ISCSI_NAME_MAX_SIZE 224

Referenced by IScsiDhcp6ExtractRootPath(), IScsiDhcpExtractRootPath(), IScsiFormCallback(), IScsiFormExtractConfig(), IScsiGetConfigData(), and IScsiSetInitiatorName().

#define ISID_CONFIGURABLE_MAX_LEN 12

#define ISID_CONFIGURABLE_MIN_LEN 6

#define ISID_CONFIGURABLE_STORAGE 13

Referenced by IScsiConvertIsIdToString().

#define KERBEROS_KDC_PORT_MAX_NUM 65535

#define KERBEROS_KDC_PORT_MIN_NUM 0

#define KERBEROS_NAME_MAX_LEN 96

#define KERBEROS_SECRET_MAX_LEN 16

#define KERBEROS_SECRET_MIN_LEN 12

#define KERBEROS_SECRET_STORAGE 17

#define KEY_ADD_ATTEMPT 0x10e

Referenced by IScsiFormCallback().

#define KEY_ATTEMPT_ENTRY_BASE 0xa000

Referenced by IScsiConfigProcessDefault(), and IScsiConfigUpdateAttempt().

#define KEY_ATTEMPT_NAME 0x113

Referenced by IScsiFormCallback().

#define KEY_AUTH_TYPE 0x11d

Referenced by IScsiFormCallback().

#define KEY_BOOT_LUN 0x10d

Referenced by IScsiFormCallback().

#define KEY_CHAP_NAME 0x107

Referenced by IScsiFormCallback().

#define KEY_CHAP_SECRET 0x108

Referenced by IScsiFormCallback().

#define KEY_CONFIG_ISID 0x11e

Referenced by IScsiFormCallback().

#define KEY_DE_ATTEMPT_ENTRY_BASE 0xb000

#define KEY_DELETE_ATTEMPT 0x116

Referenced by IScsiFormCallback().

#define KEY_DEVICE_ENTRY_BASE 0x1000

#define KEY_DHCP_ENABLE 0x102

Referenced by IScsiFormCallback().

#define KEY_GATE_WAY 0x105

Referenced by IScsiFormCallback().

#define KEY_IGNORE_DELETE_ATTEMPT 0x115

Referenced by IScsiFormCallback().

#define KEY_IGNORE_ORDER_CHANGES 0x112

Referenced by IScsiFormCallback().

#define KEY_INITIATOR_NAME 0x101

Referenced by IScsiFormCallback().

#define KEY_IP_MODE 0x11c

Referenced by IScsiFormCallback().

#define KEY_LOCAL_IP 0x103

Referenced by IScsiFormCallback().

#define KEY_MAC_ENTRY_BASE 0x2000

Referenced by IScsiConfigAddAttempt(), and IScsiConfigProcessDefault().

#define KEY_ORDER_ATTEMPT_CONFIG 0x110

Referenced by IScsiFormCallback().

#define KEY_REVERSE_CHAP_NAME 0x109

Referenced by IScsiFormCallback().

#define KEY_REVERSE_CHAP_SECRET 0x10a

Referenced by IScsiFormCallback().

#define KEY_SAVE_ATTEMPT_CONFIG 0x10f

Referenced by IScsiFormCallback().

#define KEY_SAVE_CHANGES 0x10b

#define KEY_SAVE_DELETE_ATTEMPT 0x114

Referenced by IScsiFormCallback().

#define KEY_SAVE_ORDER_CHANGES 0x111

Referenced by IScsiFormCallback().

#define KEY_SUBNET_MASK 0x104

Referenced by IScsiFormCallback().

#define KEY_TARGET_IP 0x106

Referenced by IScsiFormCallback().

#define KEY_TARGET_NAME 0x10c

Referenced by IScsiFormCallback().

#define LABEL_END 0xffff

#define LUN_MAX_SIZE 20

#define LUN_MIN_SIZE 1

#define MAC_ENTRY_LABEL 0x3000

Referenced by IScsiConfigAddAttempt().

#define ORDER_ENTRY_LABEL 0x4000

Referenced by IScsiConfigDisplayOrderAttempts().

#define TARGET_PORT_MAX_NUM 65535

#define TARGET_PORT_MIN_NUM 0

#define VAR_EQ_TEST_NAME 0x100

Typedef Documentation

typedef struct _ISCSI_CONFIG_IFR_NVDATA ISCSI_CONFIG_IFR_NVDATA

Generated on Mon Sep 28 08:49:55 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IScsiDxe/IScsiConfigVfr.vfr File Reference

Defines

"memItemRight" valign="bottom">EFI_NETWORK_DEVICE_CLASS 0x04

Variables

```
                    formset guid = ISCSI_CONFIG_GUID
                    formset title =
                            STRING_TOKEN(STR_ISCSI_CON
                    formset help =
                            STRING_TOKEN(STR_ISCSI_CON
          formset varstore ISCSI_CONFIG_IFR_NVDATA
          formset varstore varid = CONFIGURATION_VARST
          formset varstore name = ISCSI_CONFIG_IFR_NVDA
                     form formid = FORMID_MAIN_FORM
                   string prompt =
                            STRING_TOKEN(STR_ISCSI_CON
                   string flags = INTERACTIVE
                   string key = KEY_INITIATOR_NAME
                   string minsize = ISCSI_NAME_IFR_MIN_
                   string maxsize = ISCSI_NAME_IFR_MAX_
                   string endstring
                 subtitle text = STRING_TOKEN(STR_NULL
                     goto FORMID_MAC_FORM
                    label ATTEMPT_ENTRY_LABEL
                    label LABEL_END
                     goto FORMID_DELETE_FORM
                     goto FORMID_ORDER_FORM
                            endform
                    label MAC_ENTRY_LABEL
                    label ORDER_ENTRY_LABEL
                    label DELETE_ENTRY_LABEL
            oneof option value = ISCSI_DISABLED
                            endoneof
                   oneof questionid = KEY_IP_MODE
                 numeric minimum = CONNECT_MIN_RETR
                 numeric maximum = CONNECT_MAX_RET
                 numeric step = 0
                 numeric endnumeric
                 numeric default = CONNECT_DEFAULT_TI
        suppressif ideqval
ISCSI_CONFIG_IFR_NVDATA IpMode = = IP_MODE_AUTOCONF
                checkbox endcheckbox
                            endif
        suppressif ideqval
ISCSI_CONFIG_IFR_NVDATA InitiatorInfoFromDhcp
     suppressif NOT ideqval
ISCSI_CONFIG_IFR_NVDATA AuthenticationType = = ISCSI_AUT
```

goto FORMID_MAIN_FORM
endformset

Detailed Description

VFR file used by the iSCSI configuration component.

Copyright (c) 2004 - 2011, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Variable Documentation

label ATTEMPT_ENTRY_LABEL

suppressif NOT ideqval ISCSI_CONFIG_IFR_NVDATA AuthenticationType = = ISCSI_AUTH_TYPE_CHAP

numeric default = CONNECT_DEFAULT_TIMEOUT

label DELETE_ENTRY_LABEL

checkbox endcheckbox

endform

endformset

endif

numeric endnumeric

endoneof

string endstring

goto flags = INTERACTIVE

form formid = FORMID_MAIN_FORM

goto FORMID_DELETE_FORM

goto FORMID_MAC_FORM

goto FORMID_MAIN_FORM

goto FORMID_ORDER_FORM

formset varstore guid = ISCSI_CONFIG_GUID

goto help = STRING_TOKEN(STR_ISCSI_CONFIG_FORM_HELP)

suppressif ideqval ISCSI_CONFIG_IFR_NVDATA InitiatorInfoFromDhcp
Initial value:

```
= 0x01 OR
                ideqval ISCSI_CONFIG_IFR_NVDATA.IpMode == IP_MODE_IP6 OR
                ideqval ISCSI_CONFIG_IFR_NVDATA.IpMode == IP_MODE_AUTOCONFIG
```

suppressif ideqval ISCSI_CONFIG_IFR_NVDATA IpMode = = IP_MODE_AUTOCONFIG

Referenced by IScsiConvertIfrNvDataToAttemptConfigData(), IScsiDhcp6ExtractRootPath(), IScsiDhcpExtractRootPath(), IScsiGetConfigData(), and IScsiUpdateTargetAddress().

formset varstore ISCSI_CONFIG_IFR_NVDATA

text key = KEY_INITIATOR_NAME

label LABEL_END

label MAC_ENTRY_LABEL

numeric maximum = CONNECT_MAX_RETRY

string maxsize = ISCSI_NAME_IFR_MAX_SIZE

numeric minimum = CONNECT_MIN_RETRY

string minsize = ISCSI_NAME_IFR_MIN_SIZE

formset varstore name = ISCSI_CONFIG_IFR_NVDATA

label ORDER_ENTRY_LABEL

goto prompt = STRING_TOKEN(STR_ISCSI_CONFIG_INIT_NAME)

oneof questionid = KEY_IP_MODE

numeric step = 0

text text = STRING_TOKEN(STR_NULL)

form title = STRING_TOKEN(STR_ISCSI_CONFIG_FORM_TITLE)

option value = ISCSI_DISABLED

string varid = CONFIGURATION_VARSTORE_ID

NetworkPkg/IScsiDxe/IScsiDhcp.c File Reference

Functions

EFI_STATUS IScsiDhcpExtractRootPath (IN CHAR8 *RootPath, IN UINT8 Length, IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA *ConfigData)

EFI_STATUS EFIAPI IScsiDhcpSelectOffer (IN EFI_DHCP4_PROTOCOL *This, IN VOID *Context, IN EFI_DHCP4_STATE CurrentState, IN EFI_DHCP4_EVENT Dhcp4Event, IN EFI_DHCP4_PACKET *Packet, OUT EFI_DHCP4_PACKET **NewPacket)

EFI_STATUS IScsiParseDhcpAck (IN EFI_DHCP4_PROTOCOL *Dhcp4, IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA *ConfigData)

EFI_STATUS IScsiDoDhcp (IN EFI_HANDLE Image, IN EFI_HANDLE Controller, IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA *ConfigData)

Detailed Description

iSCSI DHCP4 related configuration routines.

Copyright (c) 2004 - 2011, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS IScsiDhcpExtractRootPath (  IN CHAR8 *                          RootPath,
                                       IN UINT8                            Length,
                                       IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA *  ConfigData
                                     )
```

Extract the Root Path option and get the required target information.

Parameters:

[in]	*RootPath*	The RootPath.
[in]	*Length*	Length of the RootPath option payload.
[in,out]	*ConfigData*	The iSCSI attempt configuration data read from a nonvolatile device.

Return values:

EFI_SUCCESS	All required information is extracted from the RootPath option.
EFI_NOT_FOUND	The RootPath is not an iSCSI RootPath.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_INVALID_PARAMETER	The RootPath is malformatted.

References _ISCSI_SESSION_CONFIG_NVDATA::BootLun, IP_MODE_AUTOCONFIG, _ISCSI_SESSION_CONFIG_NVDATA::IpMode, IpMode, ISCSI_NAME_MAX_SIZE, ISCSI_ROOT_PATH_FIELD_DELIMITER, ISCSI_ROOT_PATH_ID, ISCSI_WELL_KNOWN_PORT, IScsiAsciiStrToIp(), IScsiAsciiStrToLun(), IScsiNormalizeName(), _ISCSI_ROOT_PATH_FIELD::Len, RP_FIELD_IDX_LUN, RP_FIELD_IDX_MAX, RP_FIELD_IDX_PORT, RP_FIELD_IDX_PROTOCOL, RP_FIELD_IDX_SERVERNAME, RP_FIELD_IDX_TARGETNAME, _ISCSI_ROOT_PATH_FIELD::Str, _ISCSI_SESSION_CONFIG_NVDATA::TargetIp, _ISCSI_SESSION_CONFIG_NVDATA::TargetName, and _ISCSI_SESSION_CONFIG_NVDATA::TargetPort.

Referenced by IScsiDhcpSelectOffer().

```
EFI_STATUS EFIAPI IScsiDhcpSelectOffer ( IN EFI_DHCP4_PROTOCOL *   This,
                                         IN VOID *                 Context,
                                         IN EFI_DHCP4_STATE        CurrentState,
                                         IN EFI_DHCP4_EVENT        Dhcp4Event,
                                         IN EFI_DHCP4_PACKET *     Packet,
                                         OUT EFI_DHCP4_PACKET **   NewPacket
                                       )
```

The callback function registerd to the DHCP4 instance that is used to select the qualified DHCP OFFER.

Parameters:

[in]	*This*	The DHCP4 protocol.
[in]	*Context*	The context set when configuring the DHCP4 protocol.
[in]	*CurrentState*	The current state of the DHCP4 protocol.
[in]	*Dhcp4Event*	The event occurs in the current state.
[in]	*Packet*	The DHCP packet that is to be sent or was already received.
[out]	*NewPacket*	The packet used to replace the above Packet.

Return values:

EFI_SUCCESS	Either the DHCP OFFER is qualified or we're not intereseted in the Dhcp4Event.
EFI_NOT_READY	The DHCP OFFER packet doesn't match our requirements.
Others	Other errors as indicated.

References DHCP4_TAG_ROOT_PATH, and IScsiDhcpExtractRootPath().

Referenced by IScsiDoDhcp().

```
EFI_STATUS IScsiDoDhcp ( IN EFI_HANDLE                          Image,
                         IN EFI_HANDLE                          Controller,
                         IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA *   ConfigData
                       )
```

Parse the DHCP ACK to get the address configuration and DNS information.

Parameters:

[in]	*Image*	The handle of the driver image.
[in]	*Controller*	The handle of the controller.
[in,out]	*ConfigData*	The attempt configuration data.

Return values:

EFI_SUCCESS	The DNS information is got from the DHCP ACK.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_NO_MEDIA	There was a media error.
Others	Other errors as indicated.

References DHCP4_TAG_DNS, DHCP4_TAG_NETMASK, DHCP4_TAG_PARA_LIST, DHCP4_TAG_ROOT_PATH, DHCP4_TAG_ROUTER, IScsiDhcpSelectOffer(), IScsiParseDhcpAck(), and _ISCSI_SESSION_CONFIG_NVDATA::TargetInfoFromDhcp.

Referenced by IScsiGetConfigData().

```
EFI_STATUS IScsiParseDhcpAck ( IN EFI_DHCP4_PROTOCOL *                  Dhcp4,
                               IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA *     ConfigData
                             )
```

Parse the DHCP ACK to get the address configuration and DNS information.

Parameters:

`[in]`	*Dhcp4*	The DHCP4 protocol.
`[in,out]`	*ConfigData*	The session configuration data.

Return values:

EFI_SUCCESS	The DNS information is got from the DHCP ACK.
EFI_NO_MAPPING	DHCP failed to acquire address and other information.
EFI_INVALID_PARAMETER	The DHCP ACK's DNS option is malformatted.
EFI_DEVICE_ERROR	Other errors as indicated.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.

References DHCP4_TAG_DNS, DHCP4_TAG_SERVER_ID, _ISCSI_SESSION_CONFIG_NVDATA::Gateway, _ISCSI_SESSION_CONFIG_NVDATA::LocalIp, and _ISCSI_SESSION_CONFIG_NVDATA::SubnetMask.

Referenced by IScsiDoDhcp().

NetworkPkg/IScsiDxe/IScsiDhcp.h File Reference

Data Structures

struct _ISCSI_ROOT_PATH_FIELD

Defines

"memItemRight" valign="bottom">DHCP4_TAG_PARA_LIST 55

#define DHCP4_TAG_NETMASK 1
#define DHCP4_TAG_ROUTER 3
#define DHCP4_TAG_DNS 6
#define DHCP4_TAG_SERVER_ID 54
#define DHCP4_TAG_ROOT_PATH 17
#define ISCSI_ROOT_PATH_ID "iscsi:"
#define ISCSI_ROOT_PATH_FIELD_DELIMITER ':'
#define RP_FIELD_IDX_SERVERNAME 0
#define RP_FIELD_IDX_PROTOCOL 1
#define RP_FIELD_IDX_PORT 2
#define RP_FIELD_IDX_LUN 3
#define RP_FIELD_IDX_TARGETNAME 4
#define RP_FIELD_IDX_MAX 5

Typedefs

typedef struct _ISCSI_ATTEMPT_CONFIG_NVDATA ISCSI_ATTEMPT_CONFIG_NVDATA

typedef struct _ISCSI_ROOT_PATH_FIELD ISCSI_ROOT_PATH_FIELD

Functions

EFI_STATUS IScsiDoDhcp (IN EFI_HANDLE Image, IN EFI_HANDLE Controller, IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA *ConfigData)

Detailed Description

The head file of iSCSI DHCP4 related configuration routines.

Copyright (c) 2004 - 2010, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by IScsiDoDhcp(), and IScsiParseDhcpAck().

#define DHCP4_TAG_NETMASK 1

Referenced by IScsiDoDhcp().

#define DHCP4_TAG_PARA_LIST 55

#define DHCP4_TAG_ROOT_PATH 17

Referenced by IScsiDhcpSelectOffer(), and IScsiDoDhcp().

#define DHCP4_TAG_ROUTER 3

Referenced by IScsiDoDhcp().

#define DHCP4_TAG_SERVER_ID 54

Referenced by IScsiParseDhcpAck().

#define ISCSI_ROOT_PATH_FIELD_DELIMITER ':'

Referenced by IScsiDhcp6ExtractRootPath(), and IScsiDhcpExtractRootPath().

#define ISCSI_ROOT_PATH_ID "iscsi:"

Referenced by IScsiDhcp6ExtractRootPath(), and IScsiDhcpExtractRootPath().

#define RP_FIELD_IDX_LUN 3

Referenced by IScsiDhcp6ExtractRootPath(), and IScsiDhcpExtractRootPath().

#define RP_FIELD_IDX_MAX 5

Referenced by IScsiDhcp6ExtractRootPath(), and IScsiDhcpExtractRootPath().

#define RP_FIELD_IDX_PORT 2

Referenced by IScsiDhcp6ExtractRootPath(), and IScsiDhcpExtractRootPath().

#define RP_FIELD_IDX_PROTOCOL 1

Referenced by IScsiDhcp6ExtractRootPath(), and IScsiDhcpExtractRootPath().

#define RP_FIELD_IDX_SERVERNAME 0

Referenced by IScsiDhcp6ExtractRootPath(), and IScsiDhcpExtractRootPath().

#define RP_FIELD_IDX_TARGETNAME 4

Referenced by IScsiDhcp6ExtractRootPath(), and IScsiDhcpExtractRootPath().

Typedef Documentation

typedef struct _ISCSI_ATTEMPT_CONFIG_NVDATA ISCSI_ATTEMPT_CONFIG_NVDATA

typedef struct _ISCSI_ROOT_PATH_FIELD ISCSI_ROOT_PATH_FIELD

Function Documentation

```
EFI_STATUS IScsiDoDhcp ( IN EFI_HANDLE                          Image,
                         IN EFI_HANDLE                          Controller,
                         IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA *   ConfigData
                       )
```

Parse the DHCP ACK to get the address configuration and DNS information.

Parameters:

[in]	*Image*	The handle of the driver image.
[in]	*Controller*	The handle of the controller.
[in,out]	*ConfigData*	The attempt configuration data.

Return values:

EFI_SUCCESS	The DNS information is got from the DHCP ACK.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_NO_MEDIA	There was a media error.
Others	Other errors as indicated.

References DHCP4_TAG_DNS, DHCP4_TAG_NETMASK, DHCP4_TAG_PARA_LIST, DHCP4_TAG_ROOT_PATH, DHCP4_TAG_ROUTER, IScsiDhcpSelectOffer(), IScsiParseDhcpAck(), and _ISCSI_SESSION_CONFIG_NVDATA::TargetInfoFromDhcp.

Referenced by IScsiGetConfigData().

- Data Structures
- Files

- File List
- Globals

NetworkPkg/IScsiDxe/IScsiDhcp6.c File Reference

Functions

EFI_STATUS IScsiDhcp6ExtractRootPath (IN CHAR8 *RootPath, IN UINT16 Length, IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA *ConfigData)
EFI_STATUS EFIAPI IScsiDhcp6ParseReply (IN EFI_DHCP6_PROTOCOL *This, IN VOID *Context, IN EFI_DHCP6_PACKET *Packet)
EFI_STATUS IScsiDoDhcp6 (IN EFI_HANDLE Image, IN EFI_HANDLE Controller, IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA *ConfigData)

Detailed Description

iSCSI DHCP6 related configuration routines.

Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS
IScsiDhcp6ExtractRootPath ( IN CHAR8 *                          RootPath,
                            IN UINT16                           Length,
                            IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA * ConfigData
                          )
```

Extract the Root Path option and get the required target information from Boot File Uniform Resource Locator (URL) Option.

Parameters:

[in]	*RootPath*	The RootPath string.
[in]	*Length*	Length of the RootPath option payload.
[in,out]	*ConfigData*	The iSCSI session configuration data read from nonvolatile device.

Return values:

EFI_SUCCESS	All required information is extracted from the RootPath option.
EFI_NOT_FOUND	The RootPath is not an iSCSI RootPath.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_INVALID_PARAMETER	The RootPath is malformatted.

References _ISCSI_SESSION_CONFIG_NVDATA::BootLun, IP_MODE_AUTOCONFIG, _ISCSI_SESSION_CONFIG_NVDATA::IpMode, IpMode, ISCSI_NAME_MAX_SIZE,

ISCSI_ROOT_PATH_ADDR_END_DELIMITER, ISCSI_ROOT_PATH_ADDR_START_DELIMITER, ISCSI_ROOT_PATH_FIELD_DELIMITER, ISCSI_ROOT_PATH_ID, ISCSI_WELL_KNOWN_PORT, IScsiAsciiStrToIp(), IScsiAsciiStrToLun(), IScsiNormalizeName(), _ISCSI_ROOT_PATH_FIELD::Len, RP_FIELD_IDX_LUN, RP_FIELD_IDX_MAX, RP_FIELD_IDX_PORT, RP_FIELD_IDX_PROTOCOL, RP_FIELD_IDX_SERVERNAME, RP_FIELD_IDX_TARGETNAME, _ISCSI_ROOT_PATH_FIELD::Str, _ISCSI_SESSION_CONFIG_NVDATA::TargetIp, _ISCSI_SESSION_CONFIG_NVDATA::TargetName, and _ISCSI_SESSION_CONFIG_NVDATA::TargetPort.

Referenced by IScsiDhcp6ParseReply().

```
EFI_STATUS EFIAPI IScsiDhcp6ParseReply ( IN EFI_DHCP6_PROTOCOL *  This,
                                         IN VOID *                Context,
                                         IN EFI_DHCP6_PACKET *    Packet
                                       )
```

EFI_DHCP6_INFO_CALLBACK is provided by the consumer of the EFI DHCPv6 Protocol instance to intercept events that occurs in the DHCPv6 Information Request exchange process.

Parameters:

[in]	*This*	Pointer to the EFI_DHCP6_PROTOCOL instance that is used to configure this callback function.
[in]	*Context*	Pointer to the context that is initialized in the EFI_DHCP6_PROTOCOL.InfoRequest().
[in]	*Packet*	Pointer to Reply packet that has been received. The EFI DHCPv6 Protocol instance is responsible for freeing the buffer.

Return values:

EFI_SUCCESS	Tell the EFI DHCPv6 Protocol instance to finish Information Request exchange process.
EFI_NOT_READY	Tell the EFI DHCPv6 Protocol instance to continue Information Request exchange process.
EFI_ABORTED	Tell the EFI DHCPv6 Protocol instance to abort the Information Request exchange process.
EFI_UNSUPPORTED	Tell the EFI DHCPv6 Protocol instance to finish the Information Request exchange process because some request information are not received.

References DHCP6_OPT_BOOT_FILE_PARA, DHCP6_OPT_BOOT_FILE_URL, DHCP6_OPT_DNS_SERVERS, _ISCSI_ATTEMPT_CONFIG_NVDATA::DhcpServer, IScsiDhcp6ExtractRootPath(), _ISCSI_ATTEMPT_CONFIG_NVDATA::PrimaryDns, and _ISCSI_ATTEMPT_CONFIG_NVDATA::SecondaryDns.

Referenced by IScsiDoDhcp6().

```
EFI_STATUS IScsiDoDhcp6 ( IN EFI_HANDLE                          Image,
                          IN EFI_HANDLE                          Controller,
                          IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA *   ConfigData
                        )
```

Parse the DHCP ACK to get the address configuration and DNS information.

Parameters:

[in]	*Image*	The handle of the driver image.

`[in]`	*Controller*	The handle of the controller;
`[in,out]`	*ConfigData*	The attempt configuration data.

Return values:

EFI_SUCCESS	The DNS information is got from the DHCP ACK.
EFI_NO_MAPPING	DHCP failed to acquire address and other information.
EFI_INVALID_PARAMETER	The DHCP ACK's DNS option is malformatted.
EFI_DEVICE_ERROR	Some unexpected error occurred.
EFI_OUT_OF_RESOURCES	There is no sufficient resource to finish the operation.
EFI_NO_MEDIA	There was a media error.

References DHCP6_OPT_BOOT_FILE_PARA, DHCP6_OPT_BOOT_FILE_URL, DHCP6_OPT_DNS_SERVERS, DHCP6_OPT_REQUEST_OPTION, ISCSI_GET_MAPPING_TIMEOUT, and IScsiDhcp6ParseReply().

Referenced by IScsiGetConfigData().

NetworkPkg/IScsiDxe/IScsiDhcp6.h File Reference

Defines

"memItemRight" valign="bottom">DHCP6_OPT_REQUEST_OPTION 6
#define DHCP6_OPT_VENDOR_INFO 17
#define DHCP6_OPT_DNS_SERVERS 23
#define DHCP6_OPT_BOOT_FILE_URL 59
#define DHCP6_OPT_BOOT_FILE_PARA 60
#define ISCSI_ROOT_PATH_ID "iscsi:"
#define ISCSI_ROOT_PATH_FIELD_DELIMIT
#define ISCSI_ROOT_PATH_ADDR_START_D
#define ISCSI_ROOT_PATH_ADDR_END_DEL

Functions

EFI_STATUS IScsiDhcp6ExtractRootPath (IN CHAR8
UINT16 Length, IN OUT
ISCSI_ATTEMPT_CONFIG_NVDATA
EFI_STATUS IScsiDoDhcp6 (IN EFI_HANDLE Image,
EFI_HANDLE Controller, IN OUT
ISCSI_ATTEMPT_CONFIG_NVDATA

Detailed Description

The header file of iSCSI DHCP6 related configuration routines.

Copyright (c) 2004 - 2012, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by IScsiDhcp6ParseReply(), and IScsiDoDhcp6().

#define DHCP6_OPT_BOOT_FILE_URL 59

Assigned by IANA, RFC 5970

Referenced by IScsiDhcp6ParseReply(), and IScsiDoDhcp6().

#define DHCP6_OPT_DNS_SERVERS 23

Referenced by IScsiDhcp6ParseReply(), and IScsiDoDhcp6().

#define DHCP6_OPT_REQUEST_OPTION 6

Referenced by IScsiDoDhcp6().

#define DHCP6_OPT_VENDOR_INFO 17

#define ISCSI_ROOT_PATH_ADDR_END_DELIMITER ']'

Referenced by IScsiDhcp6ExtractRootPath().

#define ISCSI_ROOT_PATH_ADDR_START_DELIMITER '['

Referenced by IScsiDhcp6ExtractRootPath().

#define ISCSI_ROOT_PATH_FIELD_DELIMITER ':'

#define ISCSI_ROOT_PATH_ID "iscsi:"

Function Documentation

EFI_STATUS IScsiDhcp6ExtractRootPath	(IN CHAR8 *	*RootPath*,
	IN UINT16	*Length*,
	IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA *	*ConfigData*
	)	

Extract the Root Path option and get the required target information from Boot File Uniform Resource Locator (URL) Option.

Parameters:

[in]	*RootPath*	The RootPath string.
[in]	*Length*	Length of the RootPath option payload.
[in,out]	*ConfigData*	The iSCSI session configuration data read from nonvolatile device.

Return values:

EFI_SUCCESS	All required information is extracted from the RootPath option.
EFI_NOT_FOUND	The RootPath is not an iSCSI RootPath.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_INVALID_PARAMETER	The RootPath is malformatted.

References _ISCSI_SESSION_CONFIG_NVDATA::BootLun, IP_MODE_AUTOCONFIG, _ISCSI_SESSION_CONFIG_NVDATA::IpMode, IpMode, ISCSI_NAME_MAX_SIZE,

ISCSI_ROOT_PATH_ADDR_END_DELIMITER, ISCSI_ROOT_PATH_ADDR_START_DELIMITER, ISCSI_ROOT_PATH_FIELD_DELIMITER, ISCSI_ROOT_PATH_ID, ISCSI_WELL_KNOWN_PORT, IScsiAsciiStrToIp(), IScsiAsciiStrToLun(), IScsiNormalizeName(), _ISCSI_ROOT_PATH_FIELD::Len, RP_FIELD_IDX_LUN, RP_FIELD_IDX_MAX, RP_FIELD_IDX_PORT, RP_FIELD_IDX_PROTOCOL, RP_FIELD_IDX_SERVERNAME, RP_FIELD_IDX_TARGETNAME, _ISCSI_ROOT_PATH_FIELD::Str, _ISCSI_SESSION_CONFIG_NVDATA::TargetIp, _ISCSI_SESSION_CONFIG_NVDATA::TargetName, and _ISCSI_SESSION_CONFIG_NVDATA::TargetPort.

Referenced by IScsiDhcp6ParseReply().

```
EFI_STATUS IScsiDoDhcp6 ( IN EFI_HANDLE                          Image,
                          IN EFI_HANDLE                          Controller,
                          IN OUT ISCSI_ATTEMPT_CONFIG_NVDATA *   ConfigData
                        )
```

Parse the DHCP ACK to get the address configuration and DNS information.

Parameters:

[in]	*Image*	The handle of the driver image.
[in]	*Controller*	The handle of the controller;
[in,out]	*ConfigData*	The attempt configuration data.

Return values:

EFI_SUCCESS	The DNS information is got from the DHCP ACK.
EFI_NO_MAPPING	DHCP failed to acquire address and other information.
EFI_INVALID_PARAMETER	The DHCP ACK's DNS option is malformatted.
EFI_DEVICE_ERROR	Some unexpected error happened.
EFI_OUT_OF_RESOURCES	There is no sufficient resource to finish the operation.
EFI_NO_MEDIA	There was a media error.

Parse the DHCP ACK to get the address configuration and DNS information.

Parameters:

[in]	*Image*	The handle of the driver image.
[in]	*Controller*	The handle of the controller;
[in,out]	*ConfigData*	The attempt configuration data.

Return values:

EFI_SUCCESS	The DNS information is got from the DHCP ACK.
EFI_NO_MAPPING	DHCP failed to acquire address and other information.
EFI_INVALID_PARAMETER	The DHCP ACK's DNS option is malformatted.
EFI_DEVICE_ERROR	Some unexpected error occurred.
EFI_OUT_OF_RESOURCES	There is no sufficient resource to finish the operation.
EFI_NO_MEDIA	There was a media error.

References DHCP6_OPT_BOOT_FILE_PARA, DHCP6_OPT_BOOT_FILE_URL, DHCP6_OPT_DNS_SERVERS, DHCP6_OPT_REQUEST_OPTION, ISCSI_GET_MAPPING_TIMEOUT, and IScsiDhcp6ParseReply().

Referenced by IScsiGetConfigData().

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IScsiDxe/IScsiDriver.c File Reference

Functions

EFI_STATUS IScsiIsDevicePathSupported (IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI IScsiSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath, IN UINT8 IpVersion)

EFI_STATUS IScsiStart (IN EFI_HANDLE Image, IN EFI_HANDLE ControllerHandle, IN UINT8 IpVersion)

EFI_STATUS EFIAPI IScsiStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer, IN UINT8 IpVersion)

EFI_STATUS EFIAPI IScsiIp4DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI IScsiIp4DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI IScsiIp4DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)

EFI_STATUS EFIAPI IScsiIp6DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI IScsiIp6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI IScsiIp6DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)

EFI_STATUS EFIAPI IScsiUnload (IN EFI_HANDLE ImageHandle)

EFI_STATUS EFIAPI IScsiDriverEntryPoint (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)

Variables

EFI_DRIVER_BINDING_PROTOCOL gIScsiIp4DriverBinding

EFI_DRIVER_BINDING_PROTOCOL gIScsiIp6DriverBinding

EFI_GUID gIScsiV4PrivateGuid = ISCSI_V4_PRIVATE_GUID

EFI_GUID gIScsiV6PrivateGuid = ISCSI_V6_PRIVATE_GUID

ISCSI_PRIVATE_DATA * mPrivate = NULL

Detailed Description

The entry point of IScsi driver.

Copyright (c) 2004 - 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI IScsiDriverEntryPoint ( IN EFI_HANDLE          ImageHandle,
                                          IN EFI_SYSTEM_TABLE *  SystemTable
                                        )
```

This is the declaration of an EFI image entry point. This entry point is the same for UEFI Applications, UEFI OS Loaders, and UEFI Drivers including both device drivers and bus drivers.

The entry point for iSCSI driver which initializes the global variables and installs the driver binding, component name protocol, iSCSI initiator name protocol and Authentication Info protocol on its image.

Parameters:

[in]	*ImageHandle*	The firmware allocated handle for the UEFI image.
[in]	*SystemTable*	A pointer to the EFI System Table.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.

References ISCSI_PRIVATE_DATA::AttemptConfigs, gIScsiAuthenticationInfo, gIScsiComponentName, gIScsiComponentName2, gIScsiInitiatorName, gIScsiIp4DriverBinding, gIScsiIp6DriverBinding, IScsiConfigFormInit(), IScsiConfigFormUnload(), and ISCSI_PRIVATE_DATA::NicInfoList.

```
EFI_STATUS EFIAPI IScsiIp4DriverBindingStart ( IN EFI_DRIVER_BINDING_PROTOCOL *  This,
                                               IN EFI_HANDLE                    ControllerHandle,
                                               IN EFI_DEVICE_PATH_PROTOCOL *    RemainingDevicePath
                                             )
```

Starts a device controller or a bus controller.

The Start() function is designed to be invoked from the EFI boot service ConnectController(). As a result, much of the error checking on the parameters to Start() has been moved into this common boot service. It is legal to call Start() from other locations, but the following calling restrictions must be followed or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE. 2. If RemainingDevicePath is not NULL, then it must be a pointer to a naturally aligned EFI_DEVICE_PATH_PROTOCOL. 3. Prior to calling Start(), the Supported() function for the driver specified by This must have been called with the same calling parameters, and Supported() must have returned EFI_SUCCESS.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to start. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For a bus driver, if this parameter is NULL, then handles for all the children of Controller are created by this driver. If this parameter is not NULL and the first Device Path Node is not the End of Device Path Node, then only the handle for the child device specified by the first Device Path Node of RemainingDevicePath is created by this driver. If the first Device Path Node of RemainingDevicePath is the End of Device Path Node, no child handle is created by this driver.

Return values:

EFI_SUCCESS	The device was started.
EFI_DEVICE_ERROR	The device could not be started due to a device error. Currently not implemented.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
Others	The driver failed to start the device.

References IScsiStart().

```
EFI_STATUS EFIAPI
IScsiIp4DriverBindingStop
(
  IN EFI_DRIVER_BINDING_PROTOCOL *   This,
  IN EFI_HANDLE                      ControllerHandle,
  IN UINTN                           NumberOfChildren,
  IN EFI_HANDLE *                    ChildHandleBuffer
)
```

Stops a device controller or a bus controller.

The Stop() function is designed to be invoked from the EFI boot service DisconnectController(). As a result, much of the error checking on the parameters to Stop() has been moved into this common boot service. It is legal to call Stop() from other locations, but the following calling restrictions must be followed or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE that was used on a previous call to this same driver's Start() function. 2. The first NumberOfChildren handles of ChildHandleBuffer must all be a valid EFI_HANDLE. In addition, all of these handles must have been created in this driver's Start() function, and the Start() function must have called OpenProtocol() on ControllerHandle with an Attribute of EFI_OPEN_PROTOCOL_BY_CHILD_CONTROLLER.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
[in]	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
[in]	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References IScsiStop().

EFI_STATUS EFIAPI IScsiIp4DriverBindingSupported	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*
	)	

Tests to see if this driver supports a given controller. If a child device is provided, it tests to see if this driver supports creating a handle for the specified child device.

This function checks to see if the driver specified by This supports the device specified by ControllerHandle. Drivers typically use the device path attached to ControllerHandle and/or the services from the bus I/O abstraction attached to ControllerHandle to determine if the driver supports ControllerHandle. This function may be called many times during platform initialization. In order to reduce boot times, the tests performed by this function must be very small and take as little time as possible to execute. This function must not change the state of any hardware devices, and this function must be aware that the device specified by ControllerHandle may already be managed by the same driver or a different driver. This function must match its calls to AllocatePages() with FreePages(), AllocatePool() with FreePool(), and OpenProtocol() with CloseProtocol(). Since ControllerHandle may have been previously started by the same driver, if a protocol is already in the opened state, then it must not be closed with CloseProtocol(). This is required to guarantee the state of ControllerHandle is not modified by this function.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	The device specified by ControllerHandle and RemainingDevicePath is already managed by the driver specified by This.
EFI_ACCESS_DENIED	The device specified by ControllerHandle and RemainingDevicePath is already managed by a different driver or an application that requires exclusive access. Currently not implemented.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

References IScsiSupported().

EFI_STATUS EFIAPI IScsiIp6DriverBindingStart	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,

IN EFI_DEVICE_PATH_PROTOCOL * *RemainingDevicePath*
)

Starts a device controller or a bus controller.

The Start() function is designed to be invoked from the EFI boot service ConnectController(). As a result, much of the error checking on the parameters to Start() has been moved into this common boot service. It is legal to call Start() from other locations, but the following calling restrictions must be followed or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE. 2. If RemainingDevicePath is not NULL, then it must be a pointer to a naturally aligned EFI_DEVICE_PATH_PROTOCOL. 3. Prior to calling Start(), the Supported() function for the driver specified by This must have been called with the same calling parameters, and Supported() must have returned EFI_SUCCESS.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to start. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For a bus driver, if this parameter is NULL, then handles for all the children of Controller are created by this driver. If this parameter is not NULL and the first Device Path Node is not the End of Device Path Node, then only the handle for the child device specified by the first Device Path Node of RemainingDevicePath is created by this driver. If the first Device Path Node of RemainingDevicePath is the End of Device Path Node, no child handle is created by this driver.

Return values:

EFI_SUCCESS	The device was started.
EFI_DEVICE_ERROR	The device could not be started due to a device error. Currently not implemented.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
Others	The driver failed to start the device.

References IScsiStart().

EFI_STATUS EFIAPI IScsiIp6DriverBindingStop
(IN EFI_DRIVER_BINDING_PROTOCOL * *This*,
IN EFI_HANDLE *ControllerHandle*,
IN UINTN *NumberOfChildren*,
IN EFI_HANDLE * *ChildHandleBuffer*
)

Stops a device controller or a bus controller.

The Stop() function is designed to be invoked from the EFI boot service DisconnectController(). As a result, much of the error checking on the parameters to Stop() has been moved into this common boot service. It is legal to call Stop() from other locations, but the following calling restrictions must be followed or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE that was used on a previous call to this same driver's Start() function. 2. The first NumberOfChildren handles of ChildHandleBuffer must all be a valid EFI_HANDLE. In addition, all of these handles must have been created in this driver's Start() function, and the Start() function must have called OpenProtocol() on

ControllerHandle with an Attribute of EFI_OPEN_PROTOCOL_BY_CHILD_CONTROLLER.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
[in]	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
[in]	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References IScsiStop().

```
EFI_STATUS EFIAPI
IScsiIp6DriverBindingSupported
  ( IN EFI_DRIVER_BINDING_PROTOCOL *   This,
    IN EFI_HANDLE                      ControllerHandle,
    IN EFI_DEVICE_PATH_PROTOCOL *      RemainingDevicePath
  )
```

Tests to see if this driver supports a given controller. If a child device is provided, it tests to see if this driver supports creating a handle for the specified child device.

This function checks to see if the driver specified by This supports the device specified by ControllerHandle. Drivers typically use the device path attached to ControllerHandle and/or the services from the bus I/O abstraction attached to ControllerHandle to determine if the driver supports ControllerHandle. This function may be called many times during platform initialization. In order to reduce boot times, the tests performed by this function must be very small and take as little time as possible to execute. This function must not change the state of any hardware devices, and this function must be aware that the device specified by ControllerHandle may already be managed by the same driver or a different driver. This function must match its calls to AllocatePages() with FreePages(), AllocatePool() with FreePool(), and OpenProtocol() with CloseProtocol(). Since ControllerHandle may have been previously started by the same driver, if a protocol is already in the opened state, then it must not be closed with CloseProtocol(). This is required to guarantee the state of ControllerHandle is not modified by this function.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	

	The device specified by ControllerHandle and RemainingDevicePath is already managed by the driver specified by This.
EFI_ACCESS_DENIED	The device specified by ControllerHandle and RemainingDevicePath is already managed by a different driver or an application that requires exclusive access. Currently not implemented.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

References IScsiSupported().

```
EFI_STATUS
IScsiIsDevicePathSupported ( IN EFI_DEVICE_PATH_PROTOCOL *  RemainingDevicePath )
```

Tests to see if this driver supports the RemainingDevicePath.

Parameters:

[in] *RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The RemainingDevicePath is supported or NULL.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

Referenced by IScsiSupported().

```
EFI_STATUS IScsiStart ( IN EFI_HANDLE  Image,
                        IN EFI_HANDLE  ControllerHandle,
                        IN UINT8       IpVersion
                      )
```

Start to manage the controller. This is the worker function for IScsiIp4(6)DriverBindingStart.

Parameters:

[in] *Image*	Handle of the image.
[in] *ControllerHandle*	Handle of the controller.
[in] *IpVersion*	IP_VERSION_4 or IP_VERSION_6.

Return values:

EFI_SUCCES	This driver was started.
EFI_ALREADY_STARTED	This driver is already running on this device.
EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_NOT_FOUND	There is no sufficient information to establish the iScsi session.
EFI_DEVICE_ERROR	Failed to get TCP connection device path.

Default Tcp child

Default Tcp child

References _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptConfigIndex, ISCSI_PRIVATE_DATA::AttemptConfigs, _ISCSI_CHAP_AUTH_DATA::AuthConfig, _ISCSI_ATTEMPT_CONFIG_NVDATA::AuthConfigData, _ISCSI_SESSION::AuthData, _ISCSI_ATTEMPT_CONFIG_NVDATA::AuthenticationType, _ISCSI_SESSION::AuthType, _ISCSI_ATTEMPT_CONFIG_NVDATA::AutoConfigureMode, _ISCSI_ATTEMPT_CONFIG_NVDATA::AutoConfigureSuccess, ISCSI_PRIVATE_DATA::BootSelectedIndex, _ISCSI_ATTEMPT_CONFIG_NVDATA::CHAP, _ISCSI_SESSION::CHAP, _ISCSI_DRIVER_DATA::ChildHandle, _ISCSI_SESSION::ConfigData, _ISCSI_DRIVER_DATA::Controller, ISCSI_PRIVATE_DATA::CurrentNic, _ISCSI_DRIVER_DATA::DevicePath, _ISCSI_ATTEMPT_CONFIG_NVDATA::DhcpSuccess, _ISCSI_SESSION_CONFIG_NVDATA::Enabled, ISCSI_PRIVATE_DATA::EnableMpio, _ISCSI_DRIVER_DATA::ExtScsiPassThruHandle, gIScsiConfigGuid, gIScsiV4PrivateGuid, gIScsiV6PrivateGuid, _ISCSI_SESSION_CONFIG_NVDATA::InitiatorInfoFromDhcp, IP_MODE_AUTOCONFIG, IP_MODE_AUTOCONFIG_IP4, IP_MODE_AUTOCONFIG_IP6, IP_MODE_IP4, IP_MODE_IP6, _ISCSI_SESSION_CONFIG_NVDATA::IpMode, ISCSI_PRIVATE_DATA::Ipv6Flag, ISCSI_AUTH_TYPE_CHAP, ISCSI_AUTH_TYPE_KRB, ISCSI_CONFIG_VAR_ATTR, ISCSI_DISABLED, ISCSI_DRIVER_DATA_FROM_EXT_SCSI_PASS_THRU, ISCSI_ENABLED, ISCSI_ENABLED_FOR_MPIO, ISCSI_MAX_MAC_STRING_LEN, ISCSI_NAME_IFR_MAX_SIZE, IScsiAddNic(), IScsiCleanDriverData(), IScsiCreateDriverData(), _ISCSI_DRIVER_DATA::IScsiExtScsiPassThru, IScsiGetConfigData(), IScsiGetTcpConnDevicePath(), IScsiGetVariableAndSize(), _ISCSI_DRIVER_DATA::IScsiIdentifier, IScsiPublishIbft(), IScsiRemoveNic(), IScsiSessionAbort(), IScsiSessionInit(), IScsiSessionLogin(), IScsiSessionReLogin(), _ISCSI_ATTEMPT_CONFIG_NVDATA::MacString, _ISCSI_ATTEMPT_CONFIG_NVDATA::NicIndex, ISCSI_PRIVATE_DATA::OneSessionEstablished, ISCSI_PRIVATE_DATA::PortString, _ISCSI_SESSION::Private, _ISCSI_DRIVER_DATA::Session, _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData, _ISCSI_ATTEMPT_CONFIG_NVDATA::ValidiBFTPath, _ISCSI_ATTEMPT_CONFIG_NVDATA::ValidPath, and ISCSI_PRIVATE_DATA::ValidSinglePathCount.

Referenced by IScsiIp4DriverBindingStart(), and IScsiIp6DriverBindingStart().

```
EFI_STATUS EFIAPI IScsiStop ( IN EFI_DRIVER_BINDING_PROTOCOL *  This,
                              IN EFI_HANDLE                     ControllerHandle,
                              IN UINTN                          NumberOfChildren,
                              IN EFI_HANDLE *                   ChildHandleBuffer,
                              IN UINT8                          IpVersion
                            )
```

Stops a device controller or a bus controller. This is the worker function for IScsiIp4(6)DriverBindingStop.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
[in]	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
[in]	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.
[in]	*IpVersion*	IP_VERSION_4 or IP_VERSION_6.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References _ISCSI_DRIVER_DATA::ChildHandle, _ISCSI_SESSION::Conns, _ISCSI_DRIVER_DATA::ExtScsiPassThruHandle, gIScsiV4PrivateGuid, gIScsiV6PrivateGuid, _ISCSI_DRIVER_DATA::Image, ISCSI_DRIVER_DATA_FROM_EXT_SCSI_PASS_THRU, ISCSI_DRIVER_DATA_FROM_IDENTIFIER, IScsiCleanDriverData(), _ISCSI_DRIVER_DATA::IScsiIdentifier, IScsiPublishIbft(), IScsiRemoveNic(), IScsiSessionAbort(), _ISCSI_DRIVER_DATA::Session, and _ISCSI_CONNECTION::TcpIo.

Referenced by IScsiIp4DriverBindingStop(), and IScsiIp6DriverBindingStop().

EFI_STATUS EFIAPI IScsiSupported	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*,
	IN UINT8	*IpVersion*
	)	

Tests to see if this driver supports a given controller. This is the worker function for IScsiIp4(6)DriverBindingSupported.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.
[in]	*IpVersion*	IP_VERSION_4 or IP_VERSION_6.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by the driver specified by This.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

References gIScsiV4PrivateGuid, gIScsiV6PrivateGuid, IScsiDhcpIsConfigured(), and IScsiIsDevicePathSupported().

Referenced by IScsiIp4DriverBindingSupported(), and IScsiIp6DriverBindingSupported().

EFI_STATUS EFIAPI IScsiUnload (IN EFI_HANDLE *ImageHandle*)

Unload the iSCSI driver.

Parameters:

[in]	*ImageHandle*	The handle of the driver image.

Return values:

EFI_SUCCESS	The driver is unloaded.

EFI_DEVICE_ERROR An unexpected error occurred.

References gIScsiAuthenticationInfo, gIScsiControllerNameTable, gIScsiInitiatorName, gIScsiIp4DriverBinding, gIScsiIp6DriverBinding, IScsiConfigFormUnload(), and IScsiTestManagedDevice().

Variable Documentation

EFI_DRIVER_BINDING_PROTOCOL gIScsiIp4DriverBinding

Initial value:

```
{
 IScsiIp4DriverBindingSupported,
 IScsiIp4DriverBindingStart,
 IScsiIp4DriverBindingStop,
 0xa,
 NULL,
 NULL
}
```

Referenced by IScsiDriverEntryPoint(), and IScsiUnload().

EFI_DRIVER_BINDING_PROTOCOL gIScsiIp6DriverBinding

Initial value:

```
{
 IScsiIp6DriverBindingSupported,
 IScsiIp6DriverBindingStart,
 IScsiIp6DriverBindingStop,
 0xa,
 NULL,
 NULL
}
```

Referenced by IScsiDriverEntryPoint(), and IScsiUnload().

EFI_GUID gIScsiV4PrivateGuid = ISCSI_V4_PRIVATE_GUID

Referenced by IScsiComponentNameGetControllerName(), IScsiStart(), IScsiStop(), and IScsiSupported().

EFI_GUID gIScsiV6PrivateGuid = ISCSI_V6_PRIVATE_GUID

Referenced by IScsiComponentNameGetControllerName(), IScsiStart(), IScsiStop(), and IScsiSupported().

ISCSI_PRIVATE_DATA* mPrivate = NULL

Referenced by IScsiAddNic(), IScsiCHAPToSendReq(), IScsiCleanDriverData(), IScsiConfigAddAttempt(), IScsiConfigDeleteAttempts(), IScsiConfigDisplayDeleteAttempts(), IScsiConfigDisplayOrderAttempts(), IScsiConfigFormUnload(), IScsiConfigGetAttemptByConfigIndex(), IScsiConfigGetAttemptByNic(), IScsiConfigOrderAttempts(), IScsiConfigProcessDefault(), IScsiConfigUpdateAttempt(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiFillInitiatorSection(), IScsiFillNICAndTargetSections(), IScsiFormCallback(), IScsiGetConfigData(), IScsiGetNicInfoByIndex(), IScsiInitControlSection(), IScsiPublishIbft(), and IScsiRemoveNic().

Generated on Mon Sep 28 08:49:55 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IScsiDxe/IScsiDriver.h File Reference

Data Structures

struct ISCSI_PRIVATE_DATA
struct ISCSI_NIC_INFO
struct _ISCSI_PRIVATE_PROTOCOL

Defines

"memItemRight" valign="bottom">ISCSI_V4_PRIVATE_GUID
#define ISCSI_V6_PRIVATE_GUID
#define ISCSI_INITIATOR_NAME_VAR_NAME L"I_NA
#define IP_MODE_AUTOCONFIG_IP4 3
#define IP_MODE_AUTOCONFIG_IP6 4

Typedefs

typedef struct
_ISCSI_PRIVATE_PROTOCOL ISCSI_PRIVATE_PROTOCOL

Functions

EFI_STATUS EFIAPI IScsiIp4DriverBindingSupported (IN
EFI_DRIVER_BINDING_PROTOCOL *This, IN
EFI_HANDLE ControllerHandle, IN
EFI_DEVICE_PATH_PROTOCOL *RemainingDev
EFI_STATUS EFIAPI IScsiIp4DriverBindingStart (IN
EFI_DRIVER_BINDING_PROTOCOL *This, IN
EFI_HANDLE ControllerHandle, IN
EFI_DEVICE_PATH_PROTOCOL *RemainingDev
EFI_STATUS EFIAPI IScsiIp4DriverBindingStop (IN
EFI_DRIVER_BINDING_PROTOCOL *This, IN
EFI_HANDLE ControllerHandle, IN UINTN
NumberOfChildren, IN EFI_HANDLE *ChildHandl
EFI_STATUS EFIAPI IScsiIp6DriverBindingSupported (IN
EFI_DRIVER_BINDING_PROTOCOL *This, IN
EFI_HANDLE ControllerHandle, IN
EFI_DEVICE_PATH_PROTOCOL *RemainingDev
EFI_STATUS EFIAPI IScsiIp6DriverBindingStart (IN
EFI_DRIVER_BINDING_PROTOCOL *This, IN
EFI_HANDLE ControllerHandle, IN
EFI_DEVICE_PATH_PROTOCOL *RemainingDev
EFI_STATUS EFIAPI IScsiIp6DriverBindingStop (IN
EFI_DRIVER_BINDING_PROTOCOL *This, IN
EFI_HANDLE ControllerHandle, IN UINTN
NumberOfChildren, IN EFI_HANDLE *ChildHandl
EFI_STATUS EFIAPI IScsiComponentNameGetDriverName (IN
EFI_COMPONENT_NAME_PROTOCOL *This, IN
CHAR8 *Language, OUT CHAR16 **DriverName)

EFI_STATUS EFIAPI IScsiComponentNameGetControllerName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE ChildHandle, IN CHAR8 *Language, OUT CHAR16 **ControllerName)

EFI_STATUS EFIAPI IScsiGetInitiatorName (IN EFI_ISCSI_INITIATOR_NAME_PROTOCOL *This, IN OUT UINTN *BufferSize, OUT VOID *Buffer)

EFI_STATUS EFIAPI IScsiSetInitiatorName (IN EFI_ISCSI_INITIATOR_NAME_PROTOCOL *This, IN OUT UINTN *BufferSize, IN VOID *Buffer)

EFI_STATUS EFIAPI IScsiGetAuthenticationInfo (IN EFI_AUTHENTICATION_INFO_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, OUT VOID **Buffer)

EFI_STATUS EFIAPI IScsiSetAuthenticationInfo (IN EFI_AUTHENTICATION_INFO_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN VOID *Buffer)

EFI_STATUS EFIAPI IScsiExtScsiPassThruFunction (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *This, IN UINT8 *Target, IN UINT64 Lun, IN OUT EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET *Packet, IN EFI_EVENT Event)

EFI_STATUS EFIAPI IScsiExtScsiPassThruGetNextTargetLun (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *This, IN OUT UINT8 **Target, IN OUT UINT64 *Lun)

EFI_STATUS EFIAPI IScsiExtScsiPassThruBuildDevicePath (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *This, IN UINT8 *Target, IN UINT64 Lun, IN OUT EFI_DEVICE_PATH_PROTOCOL **DevicePath)

EFI_STATUS EFIAPI IScsiExtScsiPassThruGetTargetLun (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *This, IN EFI_DEVICE_PATH_PROTOCOL *DevicePath, OUT UINT8 **Target, OUT UINT64 *Lun)

EFI_STATUS EFIAPI IScsiExtScsiPassThruResetChannel (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *This)

EFI_STATUS EFIAPI IScsiExtScsiPassThruResetTargetLun (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *This, IN UINT8 *Target, IN UINT64 Lun)

EFI_STATUS EFIAPI IScsiExtScsiPassThruGetNextTarget (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *This, IN OUT UINT8 **Target)

Variables

EFI_COMPONENT_NAME2_PROTOCOL gIScsiComponentName2

EFI_COMPONENT_NAME_PROTOCOL gIScsiComponentName

EFI_UNICODE_STRING_TABLE * gIScsiControllerNameTable

EFI_ISCSI_INITIATOR_NAME_PROTOCOL gIScsiInitiatorName

EFI_AUTHENTICATION_INFO_PROTOCOL gIScsiAuthenticationInfo

EFI_EXT_SCSI_PASS_THRU_PROTOCOL gIScsiExtScsiPassThruProtocolTemplate

EFI_GUID gIScsiV4PrivateGuid

EFI_GUID gIScsiV6PrivateGuid

ISCSI_PRIVATE_DATA * mPrivate

Detailed Description

The header file of IScsiDriver.c.

Copyright (c) 2004 - 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by IScsiAsciiStrToIp(), IScsiGetConfigData(), and IScsiStart().

#define IP_MODE_AUTOCONFIG_IP6 4

Referenced by IScsiAsciiStrToIp(), IScsiCreateConnection(), IScsiGetConfigData(), and IScsiStart().

#define ISCSI_INITIATOR_NAME_VAR_NAME L"I_NAME"

Referenced by IScsiGetInitiatorName(), and IScsiSetInitiatorName().

#define ISCSI_V4_PRIVATE_GUID

Value:

```
{ \
    0xfa3cde4c, 0x87c2, 0x427d, { 0xae, 0xde, 0x7d, 0xd0, 0x96, 0xc8, 0x8c, 0x58 } \
  }
```

#define ISCSI_V6_PRIVATE_GUID

Value:

```
{ \
    0x28be27e5, 0x66cc, 0x4a31, { 0xa3, 0x15, 0xdb, 0x14, 0xc3, 0x74, 0x4d, 0x85 } \
  }
```

Typedef Documentation

typedef struct _ISCSI_PRIVATE_PROTOCOL ISCSI_PRIVATE_PROTOCOL

Function Documentation

EFI_STATUS EFIAPI IScsiComponentNameGetControllerName (IN EFI_COMPONENT_NAME_PROTOCOL *This*,

```
*
IN EFI_HANDLE                    ControllerHandle,
IN EFI_HANDLE                    ChildHandle,
IN CHAR8 *                       Language,
OUT CHAR16 **                    ControllerName
)
```

Retrieves a Unicode string that is the user readable name of the controller that is being managed by a driver.

This function retrieves the user readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

`[in]`	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
`[in]`	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
`[in]`	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
`[in]`	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is determined by the driver writer. Language is specified inRFC 4646 or ISO 639-2 language code format.
`[out]`	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

Retrieves a Unicode string that is the user readable name of the controller that is being managed by a driver.

This function retrieves the user readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
[in]	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language, from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL, and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gIScsiComponentName, gIScsiControllerNameTable, gIScsiV4PrivateGuid, gIScsiV6PrivateGuid, and UpdateName().

```
EFI_STATUS EFIAPI
IScsiComponentNameGetDriverName
  (
    IN EFI_COMPONENT_NAME_PROTOCOL *   This,
    IN CHAR8 *                         Language,
    OUT CHAR16 **                      DriverName
  )
```

Retrieves a Unicode string that is the user readable name of the driver.

This function retrieves the user readable name of a driver in the form of a Unicode string. If the driver specified by This has a user readable name in the language specified by Language, then a pointer to the driver name is returned in DriverName, and EFI_SUCCESS is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*DriverName*	A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by This in the language specified by Language.

Return values:

EFI_SUCCESS	The Unicode string for the Driver specified by This and the language specified by Language was returned in DriverName.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	DriverName is NULL.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gIScsiComponentName, and mIScsiDriverNameTable.

```
EFI_STATUS EFIAPI
IScsiExtScsiPassThruBuildDevicePath
  (  IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *   This,
     IN UINT8 *                             Target,
     IN UINT64                              Lun,
     IN OUT EFI_DEVICE_PATH_PROTOCOL **     DevicePath
  )
```

Allocate and build a device path node for a SCSI device on a SCSI channel.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*Target*	The Target ID of the SCSI device for which a device path node is to be allocated and built.
[in]	*Lun*	The LUN of the SCSI device for which a device path node is to be allocated and built.
[in,out]	*DevicePath*	A pointer to a single device path node that describes the SCSI device specified by Target and Lun. This function is responsible for allocating the buffer DevicePath with the boot service AllocatePool(). It is the caller's responsibility to free DevicePath when the caller is finished with DevicePath.

Return values:

EFI_SUCCESS	The device path node that describes the SCSI device specified by Target and Lun was allocated and returned in DevicePath.

EFI_NOT_FOUND	The SCSI devices specified by Target and Lun does not exist on the SCSI channel.
EFI_INVALID_PARAMETER	DevicePath is NULL.
EFI_OUT_OF_RESOURCES	There are not enough resources to allocate DevicePath.

References _ISCSI_CHAP_AUTH_DATA::AuthConfig, _ISCSI_SESSION::AuthData, _ISCSI_SESSION::AuthType, _ISCSI_SESSION_CONFIG_NVDATA::BootLun, _ISCSI_SESSION::CHAP, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPType, _ISCSI_SESSION::ConfigData, ISCSI_AUTH_TYPE_CHAP, ISCSI_AUTH_TYPE_NONE, ISCSI_CHAP_UNI, ISCSI_DRIVER_DATA_FROM_EXT_SCSI_PASS_THRU, _ISCSI_DRIVER_DATA::Session, _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData, _ISCSI_SESSION_CONFIG_NVDATA::TargetName, and _ISCSI_SESSION::TargetPortalGroupTag.

```
EFI_STATUS EFIAPI
IScsiExtScsiPassThruFunction ( IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *          This,
                               IN UINT8 *                                    Target,
                               IN UINT64                                     Lun,
                               IN OUT
                               EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET    Packet,
                               *
                               IN EFI_EVENT                                  Event
                             )
```

Sends a SCSI Request Packet to a SCSI device that is attached to the SCSI channel. This function supports both blocking I/O and nonblocking I/O. The blocking I/O functionality is required, and the nonblocking I/O functionality is optional.

Parameters:

`[in]`	*This*	A pointer to the EFI_EXT_SCSI_PASS_THRU_PROTOCOL instance.
`[in]`	*Target*	The Target is an array of size TARGET_MAX_BYTES and it represents the id of the SCSI device to send the SCSI Request Packet. Each transport driver may choose to utilize a subset of this size to suit the needs of transport target representation. For example, a Fibre Channel driver may use only 8 bytes (WWN) to represent an FC target.
`[in]`	*Lun*	The LUN of the SCSI device to send the SCSI Request Packet.
`[in,out]`	*Packet*	A pointer to the SCSI Request Packet to send to the SCSI device specified by Target and Lun.
`[in]`	*Event*	If nonblocking I/O is not supported then Event is ignored, and blocking I/O is performed. If Event is NULL, then blocking I/O is performed. If Event is not NULL and non blocking I/O is supported, then nonblocking I/O is performed, and Event will be signaled when the SCSI Request Packet completes.

Return values:

EFI_SUCCESS	The SCSI Request Packet was sent by the host. For bi-directional commands, InTransferLength bytes were transferred from InDataBuffer. For write and bi-directional commands, OutTransferLength bytes were transferred by OutDataBuffer.
EFI_BAD_BUFFER_SIZE	The SCSI Request Packet was not executed. The number of bytes that could be transferred is returned in InTransferLength. For write and bi-directional commands, OutTransferLength bytes were transferred by OutDataBuffer.
EFI_NOT_READY	

	The SCSI Request Packet could not be sent because there are too many SCSI Request Packets already queued. The caller may retry later.
EFI_DEVICE_ERROR	A device error occurred while attempting to send the SCSI Request Packet.
EFI_INVALID_PARAMETER	Target, Lun, or the contents of ScsiRequestPacket are invalid.
EFI_UNSUPPORTED	The command described by the SCSI Request Packet is not supported by the host adapter. This includes the case of Bi-directional SCSI commands not supported by the implementation. The SCSI Request Packet was not sent, so no additional status information is available.
EFI_TIMEOUT	A timeout occurred while waiting for the SCSI Request Packet to execute.

Sends a SCSI Request Packet to a SCSI device that is attached to the SCSI channel. This function supports both blocking I/O and nonblocking I/O. The blocking I/O functionality is required, and the nonblocking I/O functionality is optional.

Parameters:

`[in]`	*This*	A pointer to the EFI_EXT_SCSI_PASS_THRU_PROTOCOL instance.
`[in]`	*Target*	The Target is an array of size TARGET_MAX_BYTES and it represents the id of the SCSI device to send the SCSI Request Packet. Each transport driver may choose to utilize a subset of this size to suit the needs of transport target representation. For example, a Fibre Channel driver may use only 8 bytes (WWN) to represent an FC target.
`[in]`	*Lun*	The LUN of the SCSI device to send the SCSI Request Packet.
`[in,out]`	*Packet*	A pointer to the SCSI Request Packet to send to the SCSI device specified by Target and Lun.
`[in]`	*Event*	If nonblocking I/O is not supported then Event is ignored, and blocking I/O is performed. If Event is NULL, then blocking I/O is performed. If Event is not NULL and non blocking I/O is supported, then nonblocking I/O is performed, and Event will be signaled when the SCSI Request Packet completes.

Return values:

EFI_SUCCESS	The SCSI Request Packet was sent by the host. For bi-directional commands, InTransferLength bytes were transferred from InDataBuffer. For write and bi-directional commands, OutTransferLength bytes were transferred by OutDataBuffer.
EFI_BAD_BUFFER_SIZE	The SCSI Request Packet was not executed. The number of bytes that could be transferred is returned in InTransferLength. For write and bi-directional commands, OutTransferLength bytes were transferred by OutDataBuffer.
EFI_NOT_READY	The SCSI Request Packet could not be sent because there are too many SCSI Request Packets already queued. The caller may retry later.
EFI_DEVICE_ERROR	A device error occurred while attempting to send the SCSI Request Packet.
EFI_INVALID_PARAMETER	Target, Lun, or the contents of ScsiRequestPacket, are invalid.
EFI_UNSUPPORTED	The command described by the SCSI Request Packet is not supported by the host adapter. This includes the case of Bi-directional SCSI commands not supported by the implementation. The SCSI Request Packet was not sent, so no additional status information is available.
EFI_TIMEOUT	A timeout occurred while waiting for the SCSI Request Packet to execute.

References ISCSI_DRIVER_DATA_FROM_EXT_SCSI_PASS_THRU, IScsiExecuteScsiCommand(), IScsiSessionReinstatement(), and _ISCSI_DRIVER_DATA::Session.

EFI_STATUS EFIAPI IScsiExtScsiPassThruGetNextTarget (	IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *	*This*,
	IN OUT UINT8 **	*Target*
)		

Retrieve the list of legal Target IDs for SCSI devices on a SCSI channel.

Parameters:

`[in]`	*This*	A pointer to the EFI_EXT_SCSI_PASS_THRU_PROTOCOL instance.
`[in,out]`	*Target*	(TARGET_MAX_BYTES) of a SCSI device present on the SCSI channel. On output, a pointer to the Target ID (an array of TARGET_MAX_BYTES) of the next SCSI device present on a SCSI channel. An input value of 0xF(all bytes in the array are 0xF) in the Target array retrieves the Target ID of the first SCSI device present on a SCSI channel.

Return values:

EFI_SUCCESS	The Target ID of the next SCSI device on the SCSI channel was returned in Target.
EFI_INVALID_PARAMETER	Target or Lun is NULL.
EFI_TIMEOUT	Target array is not all 0xF, and Target was not returned on a previous call to GetNextTarget().
EFI_NOT_FOUND	There are no more SCSI devices on this SCSI channel.

EFI_STATUS EFIAPI IScsiExtScsiPassThruGetNextTargetLun (	IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *	*This*,
	IN OUT UINT8 **	*Target*,
	IN OUT UINT64 *	*Lun*
)		

Used to retrieve the list of legal Target IDs and LUNs for SCSI devices on a SCSI channel. These can either be the list SCSI devices that are actually present on the SCSI channel, or the list of legal Target Ids and LUNs for the SCSI channel. Regardless, the caller of this function must probe the Target ID and LUN returned to see if a SCSI device is actually present at that location on the SCSI channel.

Parameters:

`[in]`	*This*	The EFI_EXT_SCSI_PASS_THRU_PROTOCOL instance.
`[in,out]`	*Target*	On input, a pointer to the Target ID of a SCSI device present on the SCSI channel. On output, a pointer to the Target ID of the next SCSI device present on a SCSI channel. An input value of 0xFFFFFFFF retrieves the Target ID of the first SCSI device present on a SCSI channel.
`[in,out]`	*Lun*	On input, a pointer to the LUN of a SCSI device present on the SCSI channel. On output, a pointer to the LUN of the next SCSI device present on a SCSI channel.

Return values:

EFI_SUCCESS	The Target ID and Lun of the next SCSI device on the SCSI channel was returned in Target and Lun.
EFI_NOT_FOUND	There are no more SCSI devices on this SCSI channel.

EFI_INVALID_PARAMETER	Target is not 0xFFFFFFFF,and Target and Lun were not returned on a previous call to GetNextDevice().

Used to retrieve the list of legal Target IDs and LUNs for SCSI devices on a SCSI channel. These can either be the list SCSI devices that are actually present on the SCSI channel, or the list of legal Target Ids and LUNs for the SCSI channel. Regardless, the caller of this function must probe the Target ID and LUN returned to see if a SCSI device is actually present at that location on the SCSI channel.

Parameters:

`[in]`	*This*	The EFI_EXT_SCSI_PASS_THRU_PROTOCOL instance.
`[in,out]`	*Target*	On input, a pointer to the Target ID of a SCSI device present on the SCSI channel. On output, a pointer to the Target ID of the next SCSI device present on a SCSI channel. An input value of 0xFFFFFFFF retrieves the Target ID of the first SCSI device present on a SCSI channel.
`[in,out]`	*Lun*	On input, a pointer to the LUN of a SCSI device present on the SCSI channel. On output, a pointer to the LUN of the next SCSI device present on a SCSI channel.

Return values:

EFI_SUCCESS	The Target ID and Lun of the next SCSI device on the SCSI channel was returned in Target and Lun.
EFI_NOT_FOUND	There are no more SCSI devices on this SCSI channel.
EFI_INVALID_PARAMETER	Target is not 0xFFFFFFFF,and Target and Lun were not returned on a previous call to GetNextDevice().

References _ISCSI_SESSION_CONFIG_NVDATA::BootLun, _ISCSI_SESSION::ConfigData, ISCSI_DRIVER_DATA_FROM_EXT_SCSI_PASS_THRU, _ISCSI_DRIVER_DATA::Session, and _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData.

```
EFI_STATUS EFIAPI
IScsiExtScsiPassThruGetTargetLun
  (  IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *   This,
     IN EFI_DEVICE_PATH_PROTOCOL *          DevicePath,
     OUT UINT8 **                           Target,
     OUT UINT64 *                           Lun
  )
```

Translate a device path node to a Target ID and LUN.

Parameters:

`[in]`	*This*	Protocol instance pointer.
`[in]`	*DevicePath*	A pointer to the device path node that describes a SCSI device on the SCSI channel.
`[out]`	*Target*	A pointer to the Target ID of a SCSI device on the SCSI channel.
`[out]`	*Lun*	A pointer to the LUN of a SCSI device on the SCSI channel.

Return values:

EFI_SUCCESS	DevicePath was successfully translated to a Target ID and LUN, and they were returned in Target and Lun.
EFI_INVALID_PARAMETER	DevicePath/Target/Lun is NULL.
EFI_UNSUPPORTED	This driver does not support the device path node type in DevicePath.
EFI_NOT_FOUND	

A valid translation from DevicePath to a Target ID and LUN does not exist.

Translate a device path node to a Target ID and LUN.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*DevicePath*	A pointer to the device path node that describes a SCSI device on the SCSI channel.
[out]	*Target*	A pointer to the Target ID of a SCSI device on the SCSI channel.
[out]	*Lun*	A pointer to the LUN of a SCSI device on the SCSI channel.

Return values:

EFI_SUCCESS	DevicePath was successfully translated to a Target ID and LUN, and they were returned in Target and Lun.
EFI_INVALID_PARAMETER	DevicePath/Target/Lun is NULL.
EFI_UNSUPPORTED	This driver does not support the device path node type in DevicePath.
EFI_NOT_FOUND	A valid translation does not exist from DevicePath to a TargetID and LUN.

References _ISCSI_SESSION_CONFIG_NVDATA::BootLun, _ISCSI_SESSION::ConfigData, ISCSI_DRIVER_DATA_FROM_EXT_SCSI_PASS_THRU, _ISCSI_DRIVER_DATA::Session, _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData, and _ISCSI_SESSION_CONFIG_NVDATA::TargetName.

EFI_STATUS EFIAPI IScsiExtScsiPassThruResetChannel (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL * *This*)

Resets a SCSI channel.This operation resets all the SCSI devices connected to the SCSI channel.

Parameters:

[in] *This* Protocol instance pointer.

Return values:

EFI_UNSUPPORTED It is not supported.

Resets a SCSI channel. This operation resets all the SCSI devices connected to the SCSI channel.

Parameters:

[in] *This* Protocol instance pointer.

Return values:

EFI_UNSUPPORTED It is not supported.

EFI_STATUS EFIAPI IScsiExtScsiPassThruResetTargetLun (
IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL * *This*,
IN UINT8 * *Target*,
IN UINT64 *Lun*
)

Resets a SCSI device that is connected to a SCSI channel.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*Target*	The Target ID of the SCSI device to reset.
[in]	*Lun*	The LUN of the SCSI device to reset.

Return values:

EFI_UNSUPPORTED	It is not supported.

```
EFI_STATUS EFIAPI
IScsiGetAuthenticationInfo (
  IN EFI_AUTHENTICATION_INFO_PROTOCOL *  This,
  IN EFI_HANDLE                          ControllerHandle,
  OUT VOID **                            Buffer
)
```

Retrieves the authentication information associated with a particular controller handle.

Parameters:

[in]	*This*	Pointer to the EFI_AUTHENTICATION_INFO_PROTOCOL.
[in]	*ControllerHandle*	Handle to the Controller.
[out]	*Buffer*	Pointer to the authentication information. This function is responsible for allocating the buffer and it is the caller's responsibility to free buffer when the caller is finished with buffer.

Return values:

EFI_DEVICE_ERROR	The authentication information could not be retrieved due to a hardware error.

```
EFI_STATUS EFIAPI
IScsiGetInitiatorName (
  IN EFI_ISCSI_INITIATOR_NAME_PROTOCOL *  This,
  IN OUT UINTN *                          BufferSize,
  OUT VOID *                              Buffer
)
```

Retrieves the current set value of iSCSI Initiator Name.

Parameters:

[in]	*This*	Pointer to the EFI_ISCSI_INITIATOR_NAME_PROTOCOL instance.
[in,out]	*BufferSize*	Size of the buffer in bytes pointed to by Buffer / Actual size of the variable data buffer.
[out]	*Buffer*	Pointer to the buffer for data to be read.

Return values:

EFI_SUCCESS	Data was successfully retrieved into the provided buffer and the BufferSize was sufficient to handle the iSCSI initiator name.
EFI_BUFFER_TOO_SMALL	BufferSize is too small for the result. BufferSize will be updated with the size required to complete the request. Buffer will not be affected.
EFI_INVALID_PARAMETER	BufferSize is NULL. BufferSize and Buffer will not be affected.
EFI_INVALID_PARAMETER	Buffer is NULL. BufferSize and Buffer will not be affected.
EFI_DEVICE_ERROR	The iSCSI initiator name could not be retrieved due to a hardware error.

Retrieves the current set value of iSCSI Initiator Name.

Parameters:

[in]	*This*	Pointer to the EFI_ISCSI_INITIATOR_NAME_PROTOCOL instance.
[in,out]	*BufferSize*	Size of the buffer in bytes pointed to by Buffer / Actual size of the variable data buffer.
[out]	*Buffer*	Pointer to the buffer for data to be read. The data is a null-terminated UTF-8 encoded string. The maximum length is 223 characters, including the null-terminator.

Return values:

EFI_SUCCESS	Data was successfully retrieved into the provided buffer and the BufferSize was sufficient to handle the iSCSI initiator name.
EFI_BUFFER_TOO_SMALL	BufferSize is too small for the result. BufferSize will be updated with the size required to complete the request. Buffer will not be affected.
EFI_INVALID_PARAMETER	BufferSize is NULL. BufferSize and Buffer will not be affected.
EFI_INVALID_PARAMETER	Buffer is NULL. BufferSize and Buffer will not be affected.
EFI_DEVICE_ERROR	The iSCSI initiator name could not be retrieved due to a hardware error.

References ISCSI_INITIATOR_NAME_VAR_NAME.

```
EFI_STATUS EFIAPI
IScsiIp4DriverBindingStart (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
)
```

Starts a device controller or a bus controller.

The Start() function is designed to be invoked from the EFI boot service ConnectController(). As a result, much of the error checking on the parameters to Start() has been moved into this common boot service. It is legal to call Start() from other locations, but the following calling restrictions must be followed or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE. 2. If RemainingDevicePath is not NULL, then it must be a pointer to a naturally aligned EFI_DEVICE_PATH_PROTOCOL. 3. Prior to calling Start(), the Supported() function for the driver specified by This must have been called with the same calling parameters, and Supported() must have returned EFI_SUCCESS.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to start. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For a bus driver, if this parameter is NULL, then handles for all the children of Controller are created by this driver. If this parameter is not NULL and the first Device Path Node is not the End of Device Path Node, then only the handle for the child device specified by the first Device Path Node of RemainingDevicePath is created by this driver. If the first Device Path Node of RemainingDevicePath is the End of Device Path Node, no child handle is created by this driver.

Return values:

EFI_SUCCESS	The device was started.
EFI_DEVICE_ERROR	The device could not be started due to a device error. Currently not implemented.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
Others	The driver failed to start the device.

References IScsiStart().

```
EFI_STATUS EFIAPI
IScsiIp4DriverBindingStop (
    IN EFI_DRIVER_BINDING_PROTOCOL *   This,
    IN EFI_HANDLE                      ControllerHandle,
    IN UINTN                           NumberOfChildren,
    IN EFI_HANDLE *                    ChildHandleBuffer
)
```

Stops a device controller or a bus controller.

The Stop() function is designed to be invoked from the EFI boot service DisconnectController(). As a result, much of the error checking on the parameters to Stop() has been moved into this common boot service. It is legal to call Stop() from other locations, but the following calling restrictions must be followed or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE that was used on a previous call to this same driver's Start() function. 2. The first NumberOfChildren handles of ChildHandleBuffer must all be a valid EFI_HANDLE. In addition, all of these handles must have been created in this driver's Start() function, and the Start() function must have called OpenProtocol() on ControllerHandle with an Attribute of EFI_OPEN_PROTOCOL_BY_CHILD_CONTROLLER.

Parameters:

`[in]`	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
`[in]`	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
`[in]`	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
`[in]`	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References IScsiStop().

```
EFI_STATUS EFIAPI
IScsiIp4DriverBindingSupported (
    IN EFI_DRIVER_BINDING_PROTOCOL *   This,
    IN EFI_HANDLE                      ControllerHandle,
    IN EFI_DEVICE_PATH_PROTOCOL *      RemainingDevicePath
)
```

Tests to see if this driver supports a given controller. If a child device is provided, it tests to see if this driver supports creating a handle for the specified child device.

This function checks to see if the driver specified by This supports the device specified by ControllerHandle. Drivers typically use the device path attached to ControllerHandle and/or the services from the bus I/O abstraction attached to ControllerHandle to determine if the driver supports ControllerHandle. This function may be called many times during platform initialization. In order to reduce boot times, the tests performed by this function must be very small and take as little time as possible to execute. This function must not change the state of any hardware devices, and this function must be aware that the device specified by ControllerHandle may already be managed by the same driver or a different driver. This function must match its calls to AllocatePages() with FreePages(), AllocatePool() with FreePool(), and OpenProtocol() with CloseProtocol(). Since ControllerHandle may have been previously started by the same driver, if a protocol is already in the opened state, then it must not be closed with CloseProtocol(). This is required to guarantee the state of ControllerHandle is not modified by this function.

Parameters:

`[in]`	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
`[in]`	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
`[in]`	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	The device specified by ControllerHandle and RemainingDevicePath is already managed by the driver specified by This.
EFI_ACCESS_DENIED	The device specified by ControllerHandle and RemainingDevicePath is already managed by a different driver or an application that requires exclusive access. Currently not implemented.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

References IScsiSupported().

```
EFI_STATUS EFIAPI
IScsiIp6DriverBindingStart (
    IN EFI_DRIVER_BINDING_PROTOCOL *  This,
    IN EFI_HANDLE                     ControllerHandle,
    IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
)
```

Starts a device controller or a bus controller.

The Start() function is designed to be invoked from the EFI boot service ConnectController(). As a result, much of the error checking on the parameters to Start() has been moved into this common boot service. It is legal to call Start() from other locations, but the following calling restrictions must be followed or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE. 2. If RemainingDevicePath is not NULL, then it must be a pointer to a naturally aligned EFI_DEVICE_PATH_PROTOCOL. 3. Prior to calling Start(), the Supported() function for the driver specified by This must have been called with the same calling parameters, and Supported() must have returned EFI_SUCCESS.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to start. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For a bus driver, if this parameter is NULL, then handles for all the children of Controller are created by this driver. If this parameter is not NULL and the first Device Path Node is not the End of Device Path Node, then only the handle for the child device specified by the first Device Path Node of RemainingDevicePath is created by this driver. If the first Device Path Node of RemainingDevicePath is the End of Device Path Node, no child handle is created by this driver.

Return values:

EFI_SUCCESS	The device was started.
EFI_DEVICE_ERROR	The device could not be started due to a device error. Currently not implemented.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.
Others	The driver failed to start the device.

References IScsiStart().

```
EFI_STATUS EFIAPI
IScsiIp6DriverBindingStop
(
  IN EFI_DRIVER_BINDING_PROTOCOL *   This,
  IN EFI_HANDLE                      ControllerHandle,
  IN UINTN                           NumberOfChildren,
  IN EFI_HANDLE *                    ChildHandleBuffer
)
```

Stops a device controller or a bus controller.

The Stop() function is designed to be invoked from the EFI boot service DisconnectController(). As a result, much of the error checking on the parameters to Stop() has been moved into this common boot service. It is legal to call Stop() from other locations, but the following calling restrictions must be followed or the system behavior will not be deterministic. 1. ControllerHandle must be a valid EFI_HANDLE that was used on a previous call to this same driver's Start() function. 2. The first NumberOfChildren handles of ChildHandleBuffer must all be a valid EFI_HANDLE. In addition, all of these handles must have been created in this driver's Start() function, and the Start() function must have called OpenProtocol() on ControllerHandle with an Attribute of EFI_OPEN_PROTOCOL_BY_CHILD_CONTROLLER.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
[in]	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
[in]	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References IScsiStop().

EFI_STATUS EFIAPI IScsiIp6DriverBindingSupported	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*
	)	

Tests to see if this driver supports a given controller. If a child device is provided, it tests to see if this driver supports creating a handle for the specified child device.

This function checks to see if the driver specified by This supports the device specified by ControllerHandle. Drivers typically use the device path attached to ControllerHandle and/or the services from the bus I/O abstraction attached to ControllerHandle to determine if the driver supports ControllerHandle. This function may be called many times during platform initialization. In order to reduce boot times, the tests performed by this function must be very small and take as little time as possible to execute. This function must not change the state of any hardware devices, and this function must be aware that the device specified by ControllerHandle may already be managed by the same driver or a different driver. This function must match its calls to AllocatePages() with FreePages(), AllocatePool() with FreePool(), and OpenProtocol() with CloseProtocol(). Since ControllerHandle may have been previously started by the same driver, if a protocol is already in the opened state, then it must not be closed with CloseProtocol(). This is required to guarantee the state of ControllerHandle is not modified by this function.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	The device specified by ControllerHandle and RemainingDevicePath is already managed by the driver specified by This.
EFI_ACCESS_DENIED	The device specified by ControllerHandle and RemainingDevicePath is already managed by a different driver or an application that requires exclusive access. Currently not implemented.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

References IScsiSupported().

EFI_STATUS EFIAPI IScsiSetAuthenticationInfo	(IN EFI_AUTHENTICATION_INFO_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,

```
                    IN VOID *                    Buffer
                )
```

Set the authentication information for a given controller handle.

Parameters:

[in]	*This*	Pointer to the EFI_AUTHENTICATION_INFO_PROTOCOL.
[in]	*ControllerHandle*	Handle to the Controller.
[in]	*Buffer*	Pointer to the authentication information.

Return values:

EFI_UNSUPPORTED	If the platform policies do not allow setting of the authentication information.

```
EFI_STATUS EFIAPI
IScsiSetInitiatorName    (  IN EFI_ISCSI_INITIATOR_NAME_PROTOCOL *   This,
                            IN OUT UINTN *                           BufferSize,
                            IN VOID *                                Buffer
                         )
```

Sets the iSSI Initiator Name.

Parameters:

[in]	*This*	Pointer to the EFI_ISCSI_INITIATOR_NAME_PROTOCOL instance.
[in,out]	*BufferSize*	Size of the buffer in bytes pointed to by Buffer.
[in]	*Buffer*	Pointer to the buffer for data to be written.

Return values:

EFI_SUCCESS	Data was successfully stored by the protocol.
EFI_UNSUPPORTED	Platform policies do not allow for data to be written.
EFI_INVALID_PARAMETER	BufferSize exceeds the maximum allowed limit. BufferSize will be updated with the maximum size required to complete the request.
EFI_INVALID_PARAMETER	Buffersize is NULL. BufferSize and Buffer will not be affected.
EFI_INVALID_PARAMETER	Buffer is NULL. BufferSize and Buffer will not be affected.
EFI_DEVICE_ERROR	The data could not be stored due to a hardware error.
EFI_OUT_OF_RESOURCES	Not enough storage is available to hold the data
EFI_PROTOCOL_ERROR	Input iSCSI initiator name does not adhere to RFC 3720

Sets the iSSI Initiator Name.

Parameters:

[in]	*This*	Pointer to the EFI_ISCSI_INITIATOR_NAME_PROTOCOL instance.
[in,out]	*BufferSize*	Size of the buffer in bytes pointed to by Buffer.
[in]	*Buffer*	Pointer to the buffer for data to be written. The data is a null-terminated UTF-8 encoded string. The maximum length is 223 characters, including the null-terminator.

Return values:

EFI_SUCCESS	Data was successfully stored by the protocol.
EFI_UNSUPPORTED	Platform policies do not allow for data to be written.
EFI_INVALID_PARAMETER	BufferSize exceeds the maximum allowed limit. BufferSize will be updated with the maximum size required to complete the request.
EFI_INVALID_PARAMETER	Buffersize is NULL. BufferSize and Buffer will not be affected.

EFI_INVALID_PARAMETER	Buffer is NULL. BufferSize and Buffer will not be affected.
EFI_DEVICE_ERROR	The data could not be stored due to a hardware error.
EFI_OUT_OF_RESOURCES	Not enough storage is available to hold the data
EFI_PROTOCOL_ERROR	Input iSCSI initiator name does not adhere to RFC 3720

References ISCSI_INITIATOR_NAME_VAR_NAME, ISCSI_NAME_MAX_SIZE, and IScsiNormalizeName().

Variable Documentation

EFI_AUTHENTICATION_INFO_PROTOCOL gIScsiAuthenticationInfo

Referenced by IScsiDriverEntryPoint(), and IScsiUnload().

EFI_COMPONENT_NAME_PROTOCOL gIScsiComponentName

Referenced by IScsiComponentNameGetControllerName(), IScsiComponentNameGetDriverName(), IScsiDriverEntryPoint(), and UpdateName().

EFI_COMPONENT_NAME2_PROTOCOL gIScsiComponentName2

Referenced by IScsiDriverEntryPoint(), and UpdateName().

EFI_UNICODE_STRING_TABLE* gIScsiControllerNameTable

Referenced by IScsiComponentNameGetControllerName(), IScsiUnload(), and UpdateName().

EFI_EXT_SCSI_PASS_THRU_PROTOCOL gIScsiExtScsiPassThruProtocolTemplate

Referenced by IScsiCreateDriverData().

EFI_ISCSI_INITIATOR_NAME_PROTOCOL gIScsiInitiatorName

Referenced by IScsiDriverEntryPoint(), IScsiFormCallback(), IScsiFormExtractConfig(), IScsiGetConfigData(), and IScsiUnload().

EFI_GUID gIScsiV4PrivateGuid

Referenced by IScsiComponentNameGetControllerName(), IScsiStart(), IScsiStop(), and IScsiSupported().

EFI_GUID gIScsiV6PrivateGuid

Referenced by IScsiComponentNameGetControllerName(), IScsiStart(), IScsiStop(), and IScsiSupported().

ISCSI_PRIVATE_DATA* mPrivate

Referenced by IScsiAddNic(), IScsiCHAPToSendReq(), IScsiCleanDriverData(), IScsiConfigAddAttempt(), IScsiConfigDeleteAttempts(), IScsiConfigDisplayDeleteAttempts(), IScsiConfigDisplayOrderAttempts(), IScsiConfigFormUnload(), IScsiConfigGetAttemptByConfigIndex(), IScsiConfigGetAttemptByNic(), IScsiConfigOrderAttempts(), IScsiConfigProcessDefault(), IScsiConfigUpdateAttempt(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiFillInitiatorSection(), IScsiFillNICAndTargetSections(), IScsiFormCallback(), IScsiGetConfigData(), IScsiGetNicInfoByIndex(), IScsiInitControlSection(), IScsiPublishIbft(), and IScsiRemoveNic().

Generated on Mon Sep 28 08:49:55 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IScsiDxe/IScsiExtScsiPassThru.c File Reference

Functions

EFI_STATUS EFIAPI IScsiExtScsiPassThruFunction (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *This, IN UINT8 *Target, IN UINT64 Lun, IN OUT EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET *Packet, IN EFI_EVENT Event)

EFI_STATUS EFIAPI IScsiExtScsiPassThruGetNextTargetLun (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *This, IN OUT UINT8 **Target, IN OUT UINT64 *Lun)

EFI_STATUS EFIAPI IScsiExtScsiPassThruBuildDevicePath (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *This, IN UINT8 *Target, IN UINT64 Lun, IN OUT EFI_DEVICE_PATH_PROTOCOL **DevicePath)

EFI_STATUS EFIAPI IScsiExtScsiPassThruGetTargetLun (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *This, IN EFI_DEVICE_PATH_PROTOCOL *DevicePath, OUT UINT8 **Target, OUT UINT64 *Lun)

EFI_STATUS EFIAPI IScsiExtScsiPassThruResetChannel (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *This)

EFI_STATUS EFIAPI IScsiExtScsiPassThruResetTargetLun (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *This, IN UINT8 *Target, IN UINT64 Lun)

EFI_STATUS EFIAPI IScsiExtScsiPassThruGetNextTarget (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *This, IN OUT UINT8 **Target)

Variables

EFI_EXT_SCSI_PASS_THRU_PROTOCOL gIScsiExtScsiPassThruProtocolTemplate

---

Detailed Description

The implementation of EFI_EXT_SCSI_PASS_THRU_PROTOCOL.

Copyright (c) 2004 - 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI IScsiExtScsiPassThruBuildDevicePath	(IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *	*This*,
	IN UINT8 *	*Target*,
	IN UINT64	*Lun*,
	IN OUT EFI_DEVICE_PATH_PROTOCOL **	*DevicePath*
	)	

Allocate and build a device path node for a SCSI device on a SCSI channel.

Parameters:

`[in]`	*This*	Protocol instance pointer.
`[in]`	*Target*	The Target ID of the SCSI device for which a device path node is to be allocated and built.
`[in]`	*Lun*	The LUN of the SCSI device for which a device path node is to be allocated and built.
`[in,out]`	*DevicePath*	A pointer to a single device path node that describes the SCSI device specified by Target and Lun. This function is responsible for allocating the buffer DevicePath with the boot service AllocatePool(). It is the caller's responsibility to free DevicePath when the caller is finished with DevicePath.

Return values:

EFI_SUCCESS	The device path node that describes the SCSI device specified by Target and Lun was allocated and returned in DevicePath.
EFI_NOT_FOUND	The SCSI devices specified by Target and Lun does not exist on the SCSI channel.
EFI_INVALID_PARAMETER	DevicePath is NULL.
EFI_OUT_OF_RESOURCES	There are not enough resources to allocate DevicePath.

References _ISCSI_CHAP_AUTH_DATA::AuthConfig, _ISCSI_SESSION::AuthData, _ISCSI_SESSION::AuthType, _ISCSI_SESSION_CONFIG_NVDATA::BootLun, _ISCSI_SESSION::CHAP, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPType, _ISCSI_SESSION::ConfigData, ISCSI_AUTH_TYPE_CHAP, ISCSI_AUTH_TYPE_NONE, ISCSI_CHAP_UNI, ISCSI_DRIVER_DATA_FROM_EXT_SCSI_PASS_THRU, _ISCSI_DRIVER_DATA::Session, _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData, _ISCSI_SESSION_CONFIG_NVDATA::TargetName, and _ISCSI_SESSION::TargetPortalGroupTag.

EFI_STATUS EFIAPI IScsiExtScsiPassThruFunction	(IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *	*This*,
	IN UINT8 *	*Target*,
	IN UINT64	*Lun*,
	IN OUT EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET *	*Packet*,
	IN EFI_EVENT	*Event*
	)	

Sends a SCSI Request Packet to a SCSI device that is attached to the SCSI channel. This function supports both blocking I/O and nonblocking I/O. The blocking I/O functionality is required, and the nonblocking I/O functionality is optional.

Parameters:

[in]	*This*	A pointer to the EFI_EXT_SCSI_PASS_THRU_PROTOCOL instance.
[in]	*Target*	The Target is an array of size TARGET_MAX_BYTES and it represents the id of the SCSI device to send the SCSI Request Packet. Each transport driver may choose to utilize a subset of this size to suit the needs of transport target representation. For example, a Fibre Channel driver may use only 8 bytes (WWN) to represent an FC target.
[in]	*Lun*	The LUN of the SCSI device to send the SCSI Request Packet.
[in,out]	*Packet*	A pointer to the SCSI Request Packet to send to the SCSI device specified by Target and Lun.
[in]	*Event*	If nonblocking I/O is not supported then Event is ignored, and blocking I/O is performed. If Event is NULL, then blocking I/O is performed. If Event is not NULL and non blocking I/O is supported, then nonblocking I/O is performed, and Event will be signaled when the SCSI Request Packet completes.

Return values:

EFI_SUCCESS	The SCSI Request Packet was sent by the host. For bi-directional commands, InTransferLength bytes were transferred from InDataBuffer. For write and bi-directional commands, OutTransferLength bytes were transferred by OutDataBuffer.
EFI_BAD_BUFFER_SIZE	The SCSI Request Packet was not executed. The number of bytes that could be transferred is returned in InTransferLength. For write and bi-directional commands, OutTransferLength bytes were transferred by OutDataBuffer.
EFI_NOT_READY	The SCSI Request Packet could not be sent because there are too many SCSI Request Packets already queued. The caller may retry later.
EFI_DEVICE_ERROR	A device error occurred while attempting to send the SCSI Request Packet.
EFI_INVALID_PARAMETER	Target, Lun, or the contents of ScsiRequestPacket, are invalid.
EFI_UNSUPPORTED	The command described by the SCSI Request Packet is not supported by the host adapter. This includes the case of Bi-directional SCSI commands not supported by the implementation. The SCSI Request Packet was not sent, so no additional status information is available.
EFI_TIMEOUT	A timeout occurred while waiting for the SCSI Request Packet to execute.

References ISCSI_DRIVER_DATA_FROM_EXT_SCSI_PASS_THRU, IScsiExecuteScsiCommand(), IScsiSessionReinstatement(), and _ISCSI_DRIVER_DATA::Session.

```
EFI_STATUS EFIAPI
IScsiExtScsiPassThruGetNextTarget
(
  IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *   This,
  IN OUT UINT8 **                        Target
)
```

Retrieve the list of legal Target IDs for SCSI devices on a SCSI channel.

Parameters:

[in]	*This*	A pointer to the EFI_EXT_SCSI_PASS_THRU_PROTOCOL instance.

`[in,out]` *Target* (TARGET_MAX_BYTES) of a SCSI device present on the SCSI channel. On output, a pointer to the Target ID (an array of TARGET_MAX_BYTES) of the next SCSI device present on a SCSI channel. An input value of 0xF(all bytes in the array are 0xF) in the Target array retrieves the Target ID of the first SCSI device present on a SCSI channel.

Return values:

EFI_SUCCESS	The Target ID of the next SCSI device on the SCSI channel was returned in Target.
EFI_INVALID_PARAMETER	Target or Lun is NULL.
EFI_TIMEOUT	Target array is not all 0xF, and Target was not returned on a previous call to GetNextTarget().
EFI_NOT_FOUND	There are no more SCSI devices on this SCSI channel.

```
EFI_STATUS EFIAPI
IScsiExtScsiPassThruGetNextTargetLun
  ( IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *   This,
    IN OUT UINT8 **                        Target,
    IN OUT UINT64 *                        Lun
  )
```

Used to retrieve the list of legal Target IDs and LUNs for SCSI devices on a SCSI channel. These can either be the list SCSI devices that are actually present on the SCSI channel, or the list of legal Target Ids and LUNs for the SCSI channel. Regardless, the caller of this function must probe the Target ID and LUN returned to see if a SCSI device is actually present at that location on the SCSI channel.

Parameters:

`[in]`	*This*	The EFI_EXT_SCSI_PASS_THRU_PROTOCOL instance.
`[in,out]`	*Target*	On input, a pointer to the Target ID of a SCSI device present on the SCSI channel. On output, a pointer to the Target ID of the next SCSI device present on a SCSI channel. An input value of 0xFFFFFFFF retrieves the Target ID of the first SCSI device present on a SCSI channel.
`[in,out]`	*Lun*	On input, a pointer to the LUN of a SCSI device present on the SCSI channel. On output, a pointer to the LUN of the next SCSI device present on a SCSI channel.

Return values:

EFI_SUCCESS	The Target ID and Lun of the next SCSI device on the SCSI channel was returned in Target and Lun.
EFI_NOT_FOUND	There are no more SCSI devices on this SCSI channel.
EFI_INVALID_PARAMETER	Target is not 0xFFFFFFFF,and Target and Lun were not returned on a previous call to GetNextDevice().

References _ISCSI_SESSION_CONFIG_NVDATA::BootLun, _ISCSI_SESSION::ConfigData, ISCSI_DRIVER_DATA_FROM_EXT_SCSI_PASS_THRU, _ISCSI_DRIVER_DATA::Session, and _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData.

```
EFI_STATUS EFIAPI
IScsiExtScsiPassThruGetTargetLun
  ( IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *   This,
    IN EFI_DEVICE_PATH_PROTOCOL *          DevicePath,
```

	OUT UINT8 **	*Target*,
	OUT UINT64 *	*Lun*
)		

Translate a device path node to a Target ID and LUN.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*DevicePath*	A pointer to the device path node that describes a SCSI device on the SCSI channel.
[out]	*Target*	A pointer to the Target ID of a SCSI device on the SCSI channel.
[out]	*Lun*	A pointer to the LUN of a SCSI device on the SCSI channel.

Return values:

EFI_SUCCESS	DevicePath was successfully translated to a Target ID and LUN, and they were returned in Target and Lun.
EFI_INVALID_PARAMETER	DevicePath/Target/Lun is NULL.
EFI_UNSUPPORTED	This driver does not support the device path node type in DevicePath.
EFI_NOT_FOUND	A valid translation does not exist from DevicePath to a TargetID and LUN.

References _ISCSI_SESSION_CONFIG_NVDATA::BootLun, _ISCSI_SESSION::ConfigData, ISCSI_DRIVER_DATA_FROM_EXT_SCSI_PASS_THRU, _ISCSI_DRIVER_DATA::Session, _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData, and _ISCSI_SESSION_CONFIG_NVDATA::TargetName.

EFI_STATUS EFIAPI IScsiExtScsiPassThruResetChannel (	IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *	*This*)

Resets a SCSI channel. This operation resets all the SCSI devices connected to the SCSI channel.

Parameters:

[in]	*This*	Protocol instance pointer.

Return values:

EFI_UNSUPPORTED	It is not supported.

EFI_STATUS EFIAPI IScsiExtScsiPassThruResetTargetLun (	IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *	*This*,
	IN UINT8 *	*Target*,
	IN UINT64	*Lun*
)		

Resets a SCSI device that is connected to a SCSI channel.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*Target*	The Target ID of the SCSI device to reset.
[in]	*Lun*	The LUN of the SCSI device to reset.

Return values:

EFI_UNSUPPORTED	It is not supported.

Variable Documentation

EFI_EXT_SCSI_PASS_THRU_PROTOCOL gIScsiExtScsiPassThruProtocolTemplate

Initial value:

```
{
 NULL,
 IScsiExtScsiPassThruFunction,
 IScsiExtScsiPassThruGetNextTargetLun,
 IScsiExtScsiPassThruBuildDevicePath,
 IScsiExtScsiPassThruGetTargetLun,
 IScsiExtScsiPassThruResetChannel,
 IScsiExtScsiPassThruResetTargetLun,
 IScsiExtScsiPassThruGetNextTarget
}
```

Referenced by IScsiCreateDriverData().

NetworkPkg/IScsiDxe/IScsiIbft.c File Reference

Functions

VOID IScsiInitIbfTableHeader (OUT EFI_ACPI_ISCSI_BOOT_FIRMWARE_TABLE_HEADER *Header, IN UINT8 *OemId, IN UINT64 *OemTableId)
VOID IScsiInitControlSection (IN EFI_ACPI_ISCSI_BOOT_FIRMWARE_TABLE_HEADER *Table)
VOID IScsiAddHeapItem (IN OUT UINT8 **Heap, IN VOID *Data, IN UINTN Len)
VOID IScsiFillInitiatorSection (IN EFI_ACPI_ISCSI_BOOT_FIRMWARE_TABLE_HEADER *Table, IN OUT UINT8 **Heap)
VOID IScsiMapV4ToV6Addr (IN EFI_IPv4_ADDRESS *V4, OUT EFI_IPv6_ADDRESS *V6)
VOID IScsiFillNICAndTargetSections (IN EFI_ACPI_ISCSI_BOOT_FIRMWARE_TABLE_HEADER *Table, IN OUT UINT8 **Heap)
VOID IScsiPublishIbft (IN VOID)

Variables

BOOLEAN mIbftInstalled = FALSE
UINTN mTableKey

Detailed Description

Implementation for iSCSI Boot Firmware Table publication.

Copyright (c) 2004 - 2013, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
VOID IScsiAddHeapItem ( IN OUT UINT8 **   Heap,
                        IN VOID *         Data,
                        IN UINTN          Len
                      )
```

Add one item into the heap.

Parameters:

[in,out]	*Heap*	On input, the current address of the heap. On output, the address of the heap after the item is added.
[in]	*Data*	The data to add into the heap.
[in]	*Len*	Length of the Data in byte.

Referenced by IScsiFillInitiatorSection(), and IScsiFillNICAndTargetSections().

VOID IScsiFillInitiatorSection	(IN EFI_ACPI_ISCSI_BOOT_FIRMWARE_TABLE_HEADER *	*Table*,
	IN OUT UINT8 **	*Heap*
	)	

Fill the Initiator section of the iSCSI Boot Firmware Table.

Parameters:

[in]	*Table*	The ACPI table.
[in,out]	*Heap*	The heap.

References IBFT_ROUNDUP, ISCSI_PRIVATE_DATA::InitiatorName, ISCSI_PRIVATE_DATA::InitiatorNameLength, IScsiAddHeapItem(), and mPrivate.

Referenced by IScsiPublishIbft().

VOID IScsiFillNICAndTargetSections	(IN EFI_ACPI_ISCSI_BOOT_FIRMWARE_TABLE_HEADER *	*Table*,
	IN OUT UINT8 **	*Heap*
	)	

Fill the NIC and target sections in iSCSI Boot Firmware Table.

Parameters:

[in]	*Table*	The buffer of the ACPI table.
[in,out]	*Heap*	The heap buffer used to store the variable length parameters such as iSCSI name.

References _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptConfigIndex, ISCSI_PRIVATE_DATA::AttemptConfigs, _ISCSI_ATTEMPT_CONFIG_NVDATA::AuthConfigData, _ISCSI_ATTEMPT_CONFIG_NVDATA::AuthenticationType, _ISCSI_SESSION_CONFIG_NVDATA::BootLun, ISCSI_PRIVATE_DATA::BootSelectedIndex, ISCSI_NIC_INFO::BusNumber, _ISCSI_ATTEMPT_CONFIG_NVDATA::CHAP, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPName, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPSecret, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::CHAPType, ISCSI_NIC_INFO::DeviceNumber, _ISCSI_ATTEMPT_CONFIG_NVDATA::DhcpServer, _ISCSI_SESSION_CONFIG_NVDATA::Enabled, ISCSI_PRIVATE_DATA::EnableMpio, ISCSI_NIC_INFO::FunctionNumber, _ISCSI_SESSION_CONFIG_NVDATA::Gateway, IBFT_ROUNDUP, _ISCSI_SESSION_CONFIG_NVDATA::InitiatorInfoFromDhcp, IP_MODE_AUTOCONFIG, IP_MODE_IP4, IP_MODE_IP6, _ISCSI_SESSION_CONFIG_NVDATA::IpMode, ISCSI_AUTH_TYPE_CHAP, ISCSI_AUTH_TYPE_KRB, ISCSI_AUTH_TYPE_NONE, ISCSI_CHAP_MUTUAL, ISCSI_CHAP_UNI, ISCSI_DISABLED, ISCSI_ENABLED_FOR_MPIO, IScsiAddHeapItem(), IScsiConfigGetAttemptByConfigIndex(), IScsiGetNicInfoByIndex(), IScsiGetSubnetMaskPrefixLength(), IScsiMapV4ToV6Addr(), _ISCSI_SESSION_CONFIG_NVDATA::LocalIp, mPrivate, _ISCSI_ATTEMPT_CONFIG_NVDATA::NicIndex, ISCSI_NIC_INFO::PermanentAddress, _ISCSI_SESSION_CONFIG_NVDATA::PrefixLength, _ISCSI_ATTEMPT_CONFIG_NVDATA::PrimaryDns, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::ReverseCHAPName, _ISCSI_CHAP_AUTH_CONFIG_NVDATA::ReverseCHAPSecret,

_ISCSI_ATTEMPT_CONFIG_NVDATA::SecondaryDns, _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData, _ISCSI_SESSION_CONFIG_NVDATA::SubnetMask, _ISCSI_SESSION_CONFIG_NVDATA::TargetIp, _ISCSI_SESSION_CONFIG_NVDATA::TargetName, _ISCSI_SESSION_CONFIG_NVDATA::TargetPort, _ISCSI_ATTEMPT_CONFIG_NVDATA::ValidiBFTPath, and ISCSI_NIC_INFO::VlanId.

Referenced by IScsiPublishIbft().

VOID IScsiInitControlSection (IN EFI_ACPI_ISCSI_BOOT_FIRMWARE_TABLE_HEADER * *Table*)

Initialize the control section of the iSCSI Boot Firmware Table.

Parameters:

[in] *Table* The ACPI table.

References ISCSI_PRIVATE_DATA::EnableMpio, ISCSI_PRIVATE_DATA::Krb5MpioCount, ISCSI_PRIVATE_DATA::MpioCount, mPrivate, and ISCSI_PRIVATE_DATA::ValidSinglePathCount.

Referenced by IScsiPublishIbft().

VOID IScsiInitIbfTableHeader (OUT EFI_ACPI_ISCSI_BOOT_FIRMWARE_TABLE_HEADER * *Header*, IN UINT8 * *OemId*, IN UINT64 * *OemTableId*)

Initialize the header of the iSCSI Boot Firmware Table.

Parameters:

[out] *Header* The header of the iSCSI Boot Firmware Table.
[in] *OemId* The OEM ID.
[in] *OemTableId* The OEM table ID for the iBFT.

References IBFT_HEAP_OFFSET.

Referenced by IScsiPublishIbft().

VOID IScsiMapV4ToV6Addr (IN EFI_IPv4_ADDRESS * *V4*, OUT EFI_IPv6_ADDRESS * *V6*)

Map the v4 IP address into v6 IP address.

Parameters:

[in] *V4* The v4 IP address.
[out] *V6* The v6 IP address.

Referenced by IScsiFillNICAndTargetSections().

VOID IScsiPublishIbft (IN *VOID*)

Publish and remove the iSCSI Boot Firmware Table according to the iSCSI session status.

References ISCSI_PRIVATE_DATA::EnableMpio, IBFT_HEAP_OFFSET, IBFT_MAX_SIZE, IScsiFillInitiatorSection(), IScsiFillNICAndTargetSections(), IScsiInitControlSection(), IScsiInitIbfTableHeader(), ISCSI_PRIVATE_DATA::Krb5MpioCount, mIbftInstalled, ISCSI_PRIVATE_DATA::MpioCount, mPrivate, mTableKey, and ISCSI_PRIVATE_DATA::ValidSinglePathCount.

Referenced by IScsiStart(), and IScsiStop().

Variable Documentation

BOOLEAN mIbftInstalled = FALSE

Referenced by IScsiPublishIbft().

UINTN mTableKey

Referenced by IScsiPublishIbft().

NetworkPkg/IScsiDxe/IScsiIbft.h File Reference

Defines

"memItemRight" valign="bottom">IBFT_TABLE_VAR_NAME L"iBFT"

#define IBFT_MAX_SIZE 4096

#define IBFT_HEAP_OFFSET 2048

#define IBFT_ROUNDUP(size) NET_ROUND
EFI_ACPI_ISCSI_BOOT_FIRMWARE_

Functions

VOID IScsiPublishIbft (IN VOID)

Detailed Description

Some extra definitions for iBFT.

Copyright (c) 2004 - 2011, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by IScsiInitIbfTableHeader(), and IScsiPublishIbft().

#define IBFT_MAX_SIZE 4096

Referenced by IScsiPublishIbft().

#define IBFT_ROUNDUP (size) NET_ROUNDUP ((size), EFI_ACPI_ISCSI_BOOT_FIRMWARE_TABLE_STRUCTURE_ALIGNMENT)

Referenced by IScsiFillInitiatorSection(), and IScsiFillNICAndTargetSections().

#define IBFT_TABLE_VAR_NAME L"iBFT"

Function Documentation

VOID IScsiPublishIbft (IN *VOID*)

Publish and remove the iSCSI Boot Firmware Table according to the iSCSI session status.

References ISCSI_PRIVATE_DATA::EnableMpio, IBFT_HEAP_OFFSET, IBFT_MAX_SIZE, IScsiFillInitiatorSection(), IScsiFillNICAndTargetSections(), IScsiInitControlSection(), IScsiInitIbfTableHeader(), ISCSI_PRIVATE_DATA::Krb5MpioCount, mIbftInstalled, ISCSI_PRIVATE_DATA::MpioCount, mPrivate, mTableKey, and ISCSI_PRIVATE_DATA::ValidSinglePathCount.

Referenced by IScsiStart(), and IScsiStop().

Generated on Mon Sep 28 08:49:55 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IScsiDxe/IScsiImpl.h File Reference

Data Structures

struct _ISCSI_SESSION
struct _ISCSI_CONNECTION
struct _ISCSI_DRIVER_DATA

Defines

"memItemRight" valign="bottom">ISCSI_AUTH_INITIAL 0
#define ISCSI_SESSION_SIGNATURE SIGNATURE_32 (
#define ISCSI_GET_MAPPING_TIMEOUT 100000000U
#define ISCSI_WAIT_IPSEC_TIMEOUT 30000000U
#define ISCSI_CONNECTION_SIGNATURE SIGNATURE
'N')
#define ISCSI_DRIVER_DATA_SIGNATURE SIGNATUR
'A')
#define ISCSI_DRIVER_DATA_FROM_EXT_SCSI_PASS_
#define ISCSI_DRIVER_DATA_FROM_IDENTIFIER(Ident

Detailed Description

The shared head file for iSCSI driver.

Copyright (c) 2004 - 2012, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by IScsiCHAPOnRspReceived(), IScsiCHAPToSendReq(), IScsiCreateConnection(), and IScsiProcessLoginRsp().

#define ISCSI_CONNECTION_SIGNATURE SIGNATURE_32 ('I', 'S', 'C', 'N')

Referenced by IScsiCreateConnection(), IScsiExecuteScsiCommand(), IScsiGetTcpConnDevicePath(), and IScsiSessionAbort().

#define ISCSI_DRIVER_DATA_FROM_EXT_SCSI_PASS_THRU (PassThru)

Value:

```
CR ( \
```

```
  PassThru, \
  ISCSI_DRIVER_DATA, \
  IScsiExtScsiPassThru, \
  ISCSI_DRIVER_DATA_SIGNATURE \
  )
```

Referenced by IScsiExecuteScsiCommand(), IScsiExtScsiPassThruBuildDevicePath(), IScsiExtScsiPassThruFunction(), IScsiExtScsiPassThruGetNextTargetLun(), IScsiExtScsiPassThruGetTargetLun(), IScsiStart(), IScsiStop(), and UpdateName().

#define ISCSI_DRIVER_DATA_FROM_IDENTIFIER (Identifier)

Value:

```
CR ( \
  Identifier, \
  ISCSI_DRIVER_DATA, \
  IScsiIdentifier, \
  ISCSI_DRIVER_DATA_SIGNATURE \
  )
```

Referenced by IScsiStop().

#define ISCSI_DRIVER_DATA_SIGNATURE SIGNATURE_32 ('I', 'S', 'D', 'A')

Referenced by IScsiCreateDriverData().

#define ISCSI_GET_MAPPING_TIMEOUT 100000000U

10 seconds

Referenced by IScsiDoDhcp6().

#define ISCSI_SESSION_SIGNATURE SIGNATURE_32 ('I', 'S', 'S', 'N')

Referenced by IScsiSessionInit().

#define ISCSI_WAIT_IPSEC_TIMEOUT 30000000U

3 seconds

Referenced by IScsiSessionReLogin().

NetworkPkg/IScsiDxe/IScsiInitiatorName.c File Reference

Functions

EFI_STATUS EFIAPI IScsiGetInitiatorName (IN EFI_ISCSI_INITIATOR_NAME_PROTOCOL *This, IN OUT UINTN *BufferSize, OUT VOID *Buffer)

EFI_STATUS EFIAPI IScsiSetInitiatorName (IN EFI_ISCSI_INITIATOR_NAME_PROTOCOL *This, IN OUT UINTN *BufferSize, IN VOID *Buffer)

Variables

EFI_ISCSI_INITIATOR_NAME_PROTOCOL gIScsiInitiatorName

Detailed Description

Implementation for EFI iSCSI Initiator Name Protocol.

Copyright (c) 2004 - 2011, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI
IScsiGetInitiatorName
(
    IN EFI_ISCSI_INITIATOR_NAME_PROTOCOL *   This,
    IN OUT UINTN *                           BufferSize,
    OUT VOID *                               Buffer
)
```

Retrieves the current set value of iSCSI Initiator Name.

Parameters:

[in]	*This*	Pointer to the EFI_ISCSI_INITIATOR_NAME_PROTOCOL instance.
[in,out]	*BufferSize*	Size of the buffer in bytes pointed to by Buffer / Actual size of the variable data buffer.
[out]	*Buffer*	Pointer to the buffer for data to be read. The data is a null-terminated UTF-8 encoded string. The maximum length is 223 characters, including the null-terminator.

Return values:

EFI_SUCCESS	Data was successfully retrieved into the provided buffer and the BufferSize was sufficient to handle the iSCSI initiator name.
EFI_BUFFER_TOO_SMALL	BufferSize is too small for the result. BufferSize will be updated with the size required to complete the request. Buffer will not be affected.
EFI_INVALID_PARAMETER	BufferSize is NULL. BufferSize and Buffer will not be affected.
EFI_INVALID_PARAMETER	Buffer is NULL. BufferSize and Buffer will not be affected.
EFI_DEVICE_ERROR	The iSCSI initiator name could not be retrieved due to a hardware error.

References ISCSI_INITIATOR_NAME_VAR_NAME.

```
EFI_STATUS EFIAPI
IScsiSetInitiatorName
(
  IN EFI_ISCSI_INITIATOR_NAME_PROTOCOL *   This,
  IN OUT UINTN *                           BufferSize,
  IN VOID *                                Buffer
)
```

Sets the iSSI Initiator Name.

Parameters:

`[in]`	*This*	Pointer to the EFI_ISCSI_INITIATOR_NAME_PROTOCOL instance.
`[in,out]`	*BufferSize*	Size of the buffer in bytes pointed to by Buffer.
`[in]`	*Buffer*	Pointer to the buffer for data to be written. The data is a null-terminated UTF-8 encoded string. The maximum length is 223 characters, including the null-terminator.

Return values:

EFI_SUCCESS	Data was successfully stored by the protocol.
EFI_UNSUPPORTED	Platform policies do not allow for data to be written.
EFI_INVALID_PARAMETER	BufferSize exceeds the maximum allowed limit. BufferSize will be updated with the maximum size required to complete the request.
EFI_INVALID_PARAMETER	Buffersize is NULL. BufferSize and Buffer will not be affected.
EFI_INVALID_PARAMETER	Buffer is NULL. BufferSize and Buffer will not be affected.
EFI_DEVICE_ERROR	The data could not be stored due to a hardware error.
EFI_OUT_OF_RESOURCES	Not enough storage is available to hold the data
EFI_PROTOCOL_ERROR	Input iSCSI initiator name does not adhere to RFC 3720

References ISCSI_INITIATOR_NAME_VAR_NAME, ISCSI_NAME_MAX_SIZE, and IScsiNormalizeName().

Variable Documentation

EFI_ISCSI_INITIATOR_NAME_PROTOCOL gIScsiInitiatorName

Initial value:

```
{
  IScsiGetInitiatorName,
  IScsiSetInitiatorName
}
```

Referenced by IScsiDriverEntryPoint(), IScsiFormCallback(), IScsiFormExtractConfig(), IScsiGetConfigData(), and IScsiUnload().

Generated on Mon Sep 28 08:49:55 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/IScsiDxe/IScsiMisc.c File Reference

Functions

VOID IScsiStrTrim (IN OUT CHAR16 *Str, IN CHAR16 CharC)
UINT8 IScsiGetSubnetMaskPrefixLength (IN EFI_IPv4_ADDRESS *SubnetMask)
EFI_STATUS IScsiAsciiStrToLun (IN CHAR8 *Str, OUT UINT8 *Lun)
VOID IScsiLunToUnicodeStr (IN UINT8 *Lun, OUT CHAR16 *Str)
EFI_STATUS IScsiAsciiStrToIp (IN CHAR8 *Str, IN UINT8 IpMode, OUT EFI_IP_ADDRESS *Ip)
VOID IScsiMacAddrToStr (IN EFI_MAC_ADDRESS *Mac, IN UINT32 Len, IN UINT16 VlanId, OUT CHAR16 *Str)
EFI_STATUS IScsiBinToHex (IN UINT8 *BinBuffer, IN UINT32 BinLength, IN OUT CHAR8 *HexStr, IN OUT UINT32 *HexLength)
EFI_STATUS IScsiHexToBin (IN OUT UINT8 *BinBuffer, IN OUT UINT32 *BinLength, IN CHAR8 *HexStr)
UINTN IScsiNetNtoi (IN CHAR8 *Str)
VOID IScsiGenRandom (IN OUT UINT8 *Rand, IN UINTN RandLength)
EFI_STATUS IScsiAddNic (IN EFI_HANDLE Controller)
EFI_STATUS IScsiRemoveNic (IN EFI_HANDLE Controller)
ISCSI_NIC_INFO * IScsiGetNicInfoByIndex (IN UINT8 NicIndex)
UINT16 IScsiGetNICPciLocation (IN EFI_HANDLE Controller, OUT UINTN *Bus, OUT UINTN *Device, OUT UINTN *Function)
VOID * IScsiGetVariableAndSize (IN CHAR16 *Name, IN EFI_GUID *VendorGuid, OUT UINTN *VariableSize)
ISCSI_DRIVER_DATA * IScsiCreateDriverData (IN EFI_HANDLE Image, IN EFI_HANDLE Controller)
VOID IScsiCleanDriverData (IN ISCSI_DRIVER_DATA *Private)
BOOLEAN IScsiDhcpIsConfigured (IN EFI_HANDLE Controller, IN UINT8 IpVersion)
EFI_STATUS IScsiGetConfigData (IN ISCSI_DRIVER_DATA *Private)
EFI_DEVICE_PATH_PROTOCOL * IScsiGetTcpConnDevicePath (IN ISCSI_SESSION *Session)
VOID EFIAPI IScsiOnExitBootService (IN EFI_EVENT Event, IN VOID *Context)
EFI_STATUS EFIAPI IScsiTestManagedDevice (IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE DriverBindingHandle, IN EFI_GUID *ProtocolGuid)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED
CONST CHAR8 IScsiHexString [] = "0123456789ABCDEFabcdef"

Detailed Description

Miscellaneous routines for iSCSI driver.

Copyright (c) 2004 - 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS IScsiAddNic (IN EFI_HANDLE *Controller*)

Record the NIC info in global structure.

Parameters:

`[in]` *Controller* The handle of the controller.

Return values:

EFI_SUCCESS	The operation is completed.
EFI_OUT_OF_RESOURCES	Do not have sufficient resources to finish this operation.

References ISCSI_NIC_INFO::BusNumber, ISCSI_PRIVATE_DATA::CurrentNic, ISCSI_NIC_INFO::DeviceNumber, ISCSI_NIC_INFO::FunctionNumber, ISCSI_NIC_INFO::HwAddressSize, IScsiGetNICPciLocation(), ISCSI_NIC_INFO::Link, ISCSI_PRIVATE_DATA::MaxNic, mPrivate, ISCSI_PRIVATE_DATA::NicCount, ISCSI_NIC_INFO::NicIndex, ISCSI_PRIVATE_DATA::NicInfoList, ISCSI_NIC_INFO::PermanentAddress, and ISCSI_NIC_INFO::VlanId.

Referenced by IScsiStart().

```
EFI_STATUS IScsiAsciiStrToIp ( IN CHAR8 *                 Str,
                               IN UINT8                   IpMode,
                               OUT EFI_IP_ADDRESS *       Ip
                             )
```

Convert the formatted IP address into the binary IP address.

Parameters:

`[in]`	*Str*	The UNICODE string.
`[in]`	*IpMode*	Indicates whether the IP address is v4 or v6.
`[out]`	*Ip*	The storage to return the ASCII string.

Return values:

EFI_SUCCESS	The binary IP address is returned in Ip.
EFI_INVALID_PARAMETER	The IP string is malformatted or IpMode is invalid.

References IP_MODE_AUTOCONFIG, IP_MODE_AUTOCONFIG_IP4, IP_MODE_AUTOCONFIG_IP6, IP_MODE_IP4, and IP_MODE_IP6.

Referenced by IScsiDhcp6ExtractRootPath(), IScsiDhcpExtractRootPath(), IScsiFormCallback(), and IScsiUpdateTargetAddress().

EFI_STATUS IScsiAsciiStrToLun (IN CHAR8 * *Str*,
OUT UINT8 * *Lun*
)

Convert the hexadecimal encoded LUN string into the 64-bit LUN.

Parameters:

[in]	*Str*	The hexadecimal encoded LUN string.
[out]	*Lun*	Storage to return the 64-bit LUN.

Return values:

EFI_SUCCESS	The 64-bit LUN is stored in Lun.
EFI_INVALID_PARAMETER	The string is malformatted.

Referenced by IScsiDhcp6ExtractRootPath(), IScsiDhcpExtractRootPath(), and IScsiFormCallback().

EFI_STATUS IScsiBinToHex (IN UINT8 * *BinBuffer*,
IN UINT32 *BinLength*,
IN OUT CHAR8 * *HexStr*,
IN OUT UINT32 * *HexLength*
)

Convert the binary encoded buffer into a hexadecimal encoded string.

Parameters:

[in]	*BinBuffer*	The buffer containing the binary data.
[in]	*BinLength*	Length of the binary buffer.
[in,out]	*HexStr*	Pointer to the string.
[in,out]	*HexLength*	The length of the string.

Return values:

EFI_SUCCESS	The binary data is converted to the hexadecimal string and the length of the string is updated.
EFI_BUFFER_TOO_SMALL	The string is too small.
EFI_INVALID_PARAMETER	The IP string is malformatted.

References IScsiHexString.

Referenced by IScsiCHAPToSendReq().

VOID IScsiCleanDriverData (IN ISCSI_DRIVER_DATA * *Private*)

Clean the iSCSI driver data.

Parameters:

[in]	*Private*	The iSCSI driver data.

References mPrivate, and ISCSI_PRIVATE_DATA::OneSessionEstablished.

Referenced by IScsiStart(), and IScsiStop().

ISCSI_DRIVER_DATA* IScsiCreateDriverData (*Image*,

```
                                    IN
                                    EFI_HANDLE
                                    IN
                                    EFI_HANDLE      Controller
                                  )
```

Create the iSCSI driver data.

Parameters:

[in] *Image* The handle of the driver image.
[in] *Controller* The handle of the controller.

Returns:

The iSCSI driver data created.

Return values:

NULL Other errors as indicated.

References _ISCSI_DRIVER_DATA::Controller, _ISCSI_DRIVER_DATA::ExitBootServiceEvent, _ISCSI_DRIVER_DATA::ExtScsiPassThruHandle, _ISCSI_DRIVER_DATA::ExtScsiPassThruMode, gIScsiExtScsiPassThruProtocolTemplate, _ISCSI_DRIVER_DATA::Image, ISCSI_DRIVER_DATA_SIGNATURE, _ISCSI_DRIVER_DATA::IScsiExtScsiPassThru, IScsiOnExitBootService(), _ISCSI_DRIVER_DATA::Session, and _ISCSI_DRIVER_DATA::Signature.

Referenced by IScsiStart().

```
BOOLEAN IScsiDhcpIsConfigured ( IN EFI_HANDLE   Controller,
                                IN UINT8        IpVersion
                              )
```

Check wheather the Controller handle is configured to use DHCP protocol.

Parameters:

[in] *Controller* The handle of the controller.
[in] *IpVersion* IP_VERSION_4 or IP_VERSION_6.

Return values:

TRUE The handle of the controller need the Dhcp protocol.
FALSE The handle of the controller does not need the Dhcp protocol.

References gIScsiConfigGuid, IP_MODE_AUTOCONFIG, IP_MODE_IP4, IP_MODE_IP6, ISCSI_DISABLED, ISCSI_MAX_MAC_STRING_LEN, ISCSI_NAME_IFR_MAX_SIZE, IScsiGetVariableAndSize(), and IScsiMacAddrToStr().

Referenced by IScsiSupported().

```
VOID IScsiGenRandom ( IN OUT UINT8 *   Rand,
                      IN UINTN         RandLength
                    )
```

Generate random numbers.

Parameters:

[in,out]	*Rand*	The buffer to contain random numbers.
[in]	*RandLength*	The length of the Rand buffer.

Referenced by IScsiCHAPToSendReq().

EFI_STATUS IScsiGetConfigData (IN ISCSI_DRIVER_DATA * *Private*)

Get the various configuration data.

Parameters:

[in]	*Private*	The iSCSI driver data.

Return values:

EFI_SUCCESS	The configuration data is retrieved.
EFI_NOT_FOUND	This iSCSI driver is not configured yet.

References _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptConfigIndex, ISCSI_PRIVATE_DATA::AttemptConfigs, ISCSI_PRIVATE_DATA::AttemptCount, _ISCSI_ATTEMPT_CONFIG_NVDATA::AutoConfigureMode, _ISCSI_ATTEMPT_CONFIG_NVDATA::AutoConfigureSuccess, ISCSI_NIC_INFO::BusNumber, ISCSI_PRIVATE_DATA::CurrentNic, ISCSI_NIC_INFO::DeviceNumber, _ISCSI_ATTEMPT_CONFIG_NVDATA::DhcpSuccess, _ISCSI_SESSION_CONFIG_NVDATA::Enabled, ISCSI_PRIVATE_DATA::EnableMpio, ISCSI_NIC_INFO::FunctionNumber, gIScsiConfigGuid, gIScsiInitiatorName, ISCSI_NIC_INFO::HwAddressSize, _ISCSI_SESSION_CONFIG_NVDATA::InitiatorInfoFromDhcp, ISCSI_PRIVATE_DATA::InitiatorName, ISCSI_PRIVATE_DATA::InitiatorNameLength, IP_MODE_AUTOCONFIG, IP_MODE_AUTOCONFIG_IP4, IP_MODE_AUTOCONFIG_IP6, IP_MODE_IP4, IP_MODE_IP6, _ISCSI_SESSION_CONFIG_NVDATA::IpMode, IpMode, ISCSI_PRIVATE_DATA::Ipv6Flag, ISCSI_AUTH_TYPE_KRB, ISCSI_CONFIG_VAR_ATTR, ISCSI_DISABLED, ISCSI_ENABLED, ISCSI_ENABLED_FOR_MPIO, ISCSI_MAX_MAC_STRING_LEN, ISCSI_NAME_IFR_MAX_SIZE, ISCSI_NAME_MAX_SIZE, IScsiConfigGetAttemptByConfigIndex(), IScsiConfigUpdateAttempt(), IScsiDoDhcp(), IScsiDoDhcp6(), IScsiGetNicInfoByIndex(), IScsiGetVariableAndSize(), IScsiMacAddrToStr(), ISCSI_PRIVATE_DATA::Krb5MpioCount, _ISCSI_ATTEMPT_CONFIG_NVDATA::Link, _ISCSI_ATTEMPT_CONFIG_NVDATA::MacString, mCallbackInfo, ISCSI_PRIVATE_DATA::MpioCount, mPrivate, ISCSI_NIC_INFO::NicIndex, ISCSI_NIC_INFO::PermanentAddress, ISCSI_PRIVATE_DATA::PortString, _ISCSI_FORM_CALLBACK_INFO::RegisteredHandle, _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData, ISCSI_PRIVATE_DATA::SinglePathCount, _ISCSI_SESSION_CONFIG_NVDATA::TargetInfoFromDhcp, _ISCSI_ATTEMPT_CONFIG_NVDATA::ValidPath, ISCSI_PRIVATE_DATA::ValidSinglePathCount, and ISCSI_NIC_INFO::VlanId.

Referenced by IScsiStart().

ISCSI_NIC_INFO* IScsiGetNicInfoByIndex (IN UINT8 *NicIndex*)

Get the recorded NIC info from global structure by the Index.

Parameters:

[in]	*NicIndex*	The index indicates the position of NIC info.

Returns:

Pointer to the NIC info, or NULL if not found.

References mPrivate, ISCSI_NIC_INFO::NicIndex, and ISCSI_PRIVATE_DATA::NicInfoList.

Referenced by IScsiConfigProcessDefault(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiFillNICAndTargetSections(), and IScsiGetConfigData().

```
UINT16 IScsiGetNICPciLocation ( IN EFI_HANDLE  Controller,
                                OUT UINTN *    Bus,
                                OUT UINTN *    Device,
                                OUT UINTN *    Function
                              )
```

Get the NIC's PCI location and return it accroding to the composited format defined in iSCSI Boot Firmware Table.

Parameters:

[in]	*Controller*	The handle of the controller.
[out]	*Bus*	The bus number.
[out]	*Device*	The device number.
[out]	*Function*	The function number.

Returns:

The composited representation of the NIC PCI location.

Referenced by IScsiAddNic().

```
UINT8 IScsiGetSubnetMaskPrefixLength ( IN EFI_IPv4_ADDRESS *  SubnetMask  )
```

Calculate the prefix length of the IPv4 subnet mask.

Parameters:

[in] *SubnetMask* The IPv4 subnet mask.

Returns:

The prefix length of the subnet mask.

Return values:

0 Other errors as indicated.

Referenced by IScsiFillNICAndTargetSections(), and IScsiFormCallback().

```
EFI_DEVICE_PATH_PROTOCOL* IScsiGetTcpConnDevicePath ( IN ISCSI_SESSION *  Session  )
```

Get the device path of the iSCSI tcp connection and update it.

Parameters:

Session The iSCSI session.

Returns:

The updated device path.

Return values:

NULL Other errors as indicated.

References IP4_NODE_LEN_NEW_VERSIONS, IP6_NODE_LEN_NEW_VERSIONS, IP6_NODE_LEN_OLD_VERSIONS, IP6_OLD_IPADDRESS_OFFSET, _ISCSI_CONNECTION::Ipv6Flag, ISCSI_CONNECTION_SIGNATURE, SESSION_STATE_LOGGED_IN, and _ISCSI_CONNECTION::TcpIo.

Referenced by IScsiStart().

```
VOID* IScsiGetVariableAndSize ( IN CHAR16 *     Name,
                                IN EFI_GUID *   VendorGuid,
                                OUT UINTN *     VariableSize
                              )
```

Read the EFI variable (VendorGuid/Name) and return a dynamically allocated buffer, and the size of the buffer. If failure, return NULL.

Parameters:

[in]	*Name*	String part of EFI variable name.
[in]	*VendorGuid*	GUID part of EFI variable name.
[out]	*VariableSize*	Returns the size of the EFI variable that was read.

Returns:

Dynamically allocated memory that contains a copy of the EFI variable.

Caller is responsible freeing the buffer.

Return values:

NULL Variable was not read.

Referenced by IScsiConfigDeleteAttempts(), IScsiConfigDisplayDeleteAttempts(), IScsiConfigOrderAttempts(), IScsiConfigProcessDefault(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiDhcpIsConfigured(), IScsiGetConfigData(), and IScsiStart().

```
EFI_STATUS IScsiHexToBin ( IN OUT UINT8 *     BinBuffer,
                           IN OUT UINT32 *    BinLength,
                           IN CHAR8 *         HexStr
                         )
```

Convert the hexadecimal string into a binary encoded buffer.

Parameters:

[in,out]	*BinBuffer*	The binary buffer.
[in,out]	*BinLength*	Length of the binary buffer.
[in]	*HexStr*	The hexadecimal string.

Return values:

EFI_SUCCESS	The hexadecimal string is converted into a binary encoded buffer.
EFI_BUFFER_TOO_SMALL	The binary buffer is too small to hold the converted data.

Referenced by IScsiCHAPOnRspReceived().

```
VOID IScsiLunToUnicodeStr ( IN UINT8 *     Lun,
```

```
                                OUT CHAR16 *    Str
                              )
```

Convert the 64-bit LUN into the hexadecimal encoded LUN string.

Parameters:

[in]	*Lun*	The 64-bit LUN.
[out]	*Str*	The storage to return the hexadecimal encoded LUN string.

References IScsiHexString, and IScsiStrTrim().

Referenced by IScsiConvertAttemptConfigDataToIfrNvData().

```
VOID IScsiMacAddrToStr ( IN EFI_MAC_ADDRESS *   Mac,
                         IN UINT32              Len,
                         IN UINT16              VlanId,
                         OUT CHAR16 *           Str
                       )
```

Convert the mac address into a hexadecimal encoded "-" seperated string.

Parameters:

[in]	*Mac*	The mac address.
[in]	*Len*	Length in bytes of the mac address.
[in]	*VlanId*	VLAN ID of the network device.
[out]	*Str*	The storage to return the mac string.

References IScsiHexString.

Referenced by IScsiConfigAddAttempt(), IScsiConfigProcessDefault(), IScsiDhcpIsConfigured(), and IScsiGetConfigData().

```
UINTN IScsiNetNtoi ( IN CHAR8 *   Str   )
```

Convert the decimal-constant string or hex-constant string into a numerical value.

Parameters:

[in]	*Str*	String in decimal or hex.

Returns:

The numerical value.

Referenced by IScsiCHAPOnRspReceived(), and IScsiCheckOpParams().

```
VOID EFIAPI IScsiOnExitBootService ( IN EFI_EVENT   Event,
                                     IN VOID *      Context
                                   )
```

Abort the session when the transition from BS to RT is initiated.

Parameters:

[in]	*Event*	The event signaled.
[in]	*Context*	The iSCSI driver data.

References _ISCSI_DRIVER_DATA::ExitBootServiceEvent, IScsiSessionAbort(), and _ISCSI_DRIVER_DATA::Session.

Referenced by IScsiCreateDriverData().

EFI_STATUS IScsiRemoveNic (IN EFI_HANDLE *Controller*)

Delete the recorded NIC info from global structure. Also delete corresponding attempts.

Parameters:

`[in]` *Controller* The handle of the controller.

Return values:

EFI_SUCCESS The operation is completed.

EFI_NOT_FOUND The NIC info to be deleted is not recorded.

References ISCSI_PRIVATE_DATA::AttemptConfigs, ISCSI_PRIVATE_DATA::AttemptCount, _ISCSI_ATTEMPT_CONFIG_NVDATA::AuthenticationType, ISCSI_PRIVATE_DATA::CurrentNic, _ISCSI_SESSION_CONFIG_NVDATA::Enabled, ISCSI_PRIVATE_DATA::EnableMpio, ISCSI_NIC_INFO::HwAddressSize, ISCSI_AUTH_TYPE_KRB, ISCSI_ENABLED, ISCSI_ENABLED_FOR_MPIO, ISCSI_PRIVATE_DATA::Krb5MpioCount, _ISCSI_ATTEMPT_CONFIG_NVDATA::Link, ISCSI_NIC_INFO::Link, ISCSI_PRIVATE_DATA::MpioCount, mPrivate, ISCSI_PRIVATE_DATA::NewAttempt, ISCSI_PRIVATE_DATA::NicCount, _ISCSI_ATTEMPT_CONFIG_NVDATA::NicIndex, ISCSI_NIC_INFO::NicIndex, ISCSI_PRIVATE_DATA::NicInfoList, ISCSI_NIC_INFO::PermanentAddress, _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData, ISCSI_PRIVATE_DATA::SinglePathCount, ISCSI_PRIVATE_DATA::ValidSinglePathCount, and ISCSI_NIC_INFO::VlanId.

Referenced by IScsiStart(), and IScsiStop().

VOID IScsiStrTrim (IN OUT CHAR16 * *Str*,
IN CHAR16 *CharC*
)

Removes (trims) specified leading and trailing characters from a string.

Parameters:

`[in,out]` *Str* Pointer to the null-terminated string to be trimmed. On return, Str will hold the trimmed string.

`[in]` *CharC* Character will be trimmed from str.

Referenced by IScsiLunToUnicodeStr().

EFI_STATUS EFIAPI IScsiTestManagedDevice (IN EFI_HANDLE *ControllerHandle*,
IN EFI_HANDLE *DriverBindingHandle*,
IN EFI_GUID * *ProtocolGuid*
)

Tests whether a controller handle is being managed by IScsi driver.

This function tests whether the driver specified by DriverBindingHandle is currently managing the controller specified by ControllerHandle. This test is performed by evaluating if the the protocol specified by

ProtocolGuid is present on ControllerHandle and is was opened by DriverBindingHandle and Nic Device handle with an attribute of EFI_OPEN_PROTOCOL_BY_DRIVER. If ProtocolGuid is NULL, then ASSERT().

Parameters:

ControllerHandle	A handle for a controller to test.
DriverBindingHandle	Specifies the driver binding handle for the driver.
ProtocolGuid	Specifies the protocol that the driver specified by DriverBindingHandle opens in its Start() function.

Return values:

EFI_SUCCESS	ControllerHandle is managed by the driver specified by DriverBindingHandle.
EFI_UNSUPPORTED	ControllerHandle is not managed by the driver specified by DriverBindingHandle.

Referenced by IScsiUnload().

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED CONST CHAR8 IScsiHexString[] = "0123456789ABCDEFabcdef"

Referenced by IScsiBinToHex(), IScsiLunToUnicodeStr(), and IScsiMacAddrToStr().

NetworkPkg/IScsiDxe/IScsiMisc.h File Reference

Data Structures

struct _ISCSI_SESSION_CONFIG_NVDAT

Defines

"memItemRight" valign="bottom">IP4_NODE_LEN_NEW_VERSIONS 27

#define IP6_NODE_LEN_OLD_VERSIONS

#define IP6_NODE_LEN_NEW_VERSIONS

#define IP6_OLD_IPADDRESS_OFFSET 42

Typedefs

typedef struct _ISCSI_DRIVER_DATA ISCSI_DRIVER_DATA

typedef struct _ISCSI_SESSION_CONFIG_NVDATA ISCSI_SESSION_CONFIG_NVDATA

Functions

UINT8 IScsiGetSubnetMaskPrefixLength (IN EFI_IPv4_ADDRESS *SubnetMask)

EFI_STATUS IScsiAsciiStrToLun (IN CHAR8 *Str, OUT UINT8 *Lun)

VOID IScsiLunToUnicodeStr (IN UINT8 *Lu OUT CHAR16 *String)

VOID IScsiMacAddrToStr (IN EFI_MAC_ADDRESS *Mac, IN UINT Len, IN UINT16 VlanId, OUT CHAR1 *Str)

EFI_STATUS IScsiAsciiStrToIp (IN CHAR8 *Str, IN UINT8 IpMode, OUT EFI_IP_ADDRE *Ip)

EFI_STATUS IScsiBinToHex (IN UINT8 *BinBuffer IN UINT32 BinLength, IN OUT CHAR *HexStr, IN OUT UINT32 *HexLengt

EFI_STATUS IScsiHexToBin (IN OUT UINT8 *BinBuffer, IN OUT UINT32 *BinLength, IN CHAR8 *HexStr)

UINTN IScsiNetNtoi (IN CHAR8 *Str)

VOID IScsiGenRandom (IN OUT UINT8 *Rand, IN UINTN RandLength)

EFI_STATUS IScsiAddNic (IN EFI_HANDLE Controller)

EFI_STATUS IScsiRemoveNic (IN EFI_HANDLE Controller)

ISCSI_NIC_INFO * IScsiGetNicInfoByIndex (IN UINT8 NicIndex)

UINT16 IScsiGetNICPciLocation (IN

	EFI_HANDLE Controller, OUT UINTN *Bus, OUT UINTN *Device, OUT UINTN *Function)
VOID *	IScsiGetVariableAndSize (IN CHAR16 *Name, IN EFI_GUID *VendorGuid, OUT UINTN *VariableSize)
ISCSI_DRIVER_DATA *	IScsiCreateDriverData (IN EFI_HANDLE Image, IN EFI_HANDLE Controller)
VOID	IScsiCleanDriverData (IN ISCSI_DRIVER_DATA *Private)
BOOLEAN	IScsiDhcpIsConfigured (IN EFI_HANDLE Controller, IN UINT8 IpVersion)
EFI_STATUS	IScsiGetConfigData (IN ISCSI_DRIVER_DATA *Private)
EFI_DEVICE_PATH_PROTOCOL *	IScsiGetTcpConnDevicePath (IN ISCSI_SESSION *Session)
VOID EFIAPI	IScsiOnExitBootService (IN EFI_EVENT Event, IN VOID *Context)
EFI_STATUS EFIAPI	IScsiTestManagedDevice (IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE DriverBindingHandle, IN EFI_GUID *ProtocolGuid)

Detailed Description

Miscellaneous definitions for iSCSI driver.

Copyright (c) 2004 - 2015, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

IPv4 Device Path Node Length

Referenced by IScsiGetTcpConnDevicePath().

#define IP6_NODE_LEN_NEW_VERSIONS 60

Referenced by IScsiGetTcpConnDevicePath().

#define IP6_NODE_LEN_OLD_VERSIONS 43

IPv6 Device Path Node Length

Referenced by IScsiGetTcpConnDevicePath().

#define IP6_OLD_IPADDRESS_OFFSET 42

The ignored field StaticIpAddress's offset in old IPv6 Device Path

Referenced by IScsiGetTcpConnDevicePath().

Typedef Documentation

typedef struct _ISCSI_DRIVER_DATA ISCSI_DRIVER_DATA

typedef struct _ISCSI_SESSION_CONFIG_NVDATA ISCSI_SESSION_CONFIG_NVDATA

Function Documentation

EFI_STATUS IScsiAddNic (IN EFI_HANDLE *Controller*)

Record the NIC information in a global structure.

Parameters:

[in] *Controller* The handle of the controller.

Return values:

EFI_SUCCESS	The operation is completed.
EFI_OUT_OF_RESOURCES	Do not have sufficient resource to finish this operation.

Record the NIC info in global structure.

Parameters:

[in] *Controller* The handle of the controller.

Return values:

EFI_SUCCESS	The operation is completed.
EFI_OUT_OF_RESOURCES	Do not have sufficient resources to finish this operation.

References ISCSI_NIC_INFO::BusNumber, ISCSI_PRIVATE_DATA::CurrentNic, ISCSI_NIC_INFO::DeviceNumber, ISCSI_NIC_INFO::FunctionNumber, ISCSI_NIC_INFO::HwAddressSize, IScsiGetNICPciLocation(), ISCSI_NIC_INFO::Link, ISCSI_PRIVATE_DATA::MaxNic, mPrivate, ISCSI_PRIVATE_DATA::NicCount, ISCSI_NIC_INFO::NicIndex, ISCSI_PRIVATE_DATA::NicInfoList, ISCSI_NIC_INFO::PermanentAddress, and ISCSI_NIC_INFO::VlanId.

Referenced by IScsiStart().

EFI_STATUS IScsiAsciiStrToIp (IN CHAR8 * *Str*,
IN UINT8 *IpMode*,
OUT EFI_IP_ADDRESS * *Ip*

)

Convert the formatted IP address into the binary IP address.

Parameters:

[in]	*Str*	The UNICODE string.
[in]	*IpMode*	Indicates whether the IP address is v4 or v6.
[out]	*Ip*	The storage to return the ASCII string.

Return values:

EFI_SUCCESS	The binary IP address is returned in Ip.
EFI_INVALID_PARAMETER	The IP string is malformatted or IpMode is invalid.

References IP_MODE_AUTOCONFIG, IP_MODE_AUTOCONFIG_IP4, IP_MODE_AUTOCONFIG_IP6, IP_MODE_IP4, and IP_MODE_IP6.

Referenced by IScsiDhcp6ExtractRootPath(), IScsiDhcpExtractRootPath(), IScsiFormCallback(), and IScsiUpdateTargetAddress().

```
EFI_STATUS IScsiAsciiStrToLun ( IN CHAR8 *     Str,
                                OUT UINT8 *    Lun
                              )
```

Convert the hexadecimal encoded LUN string into the 64-bit LUN.

Parameters:

[in]	*Str*	The hexadecimal encoded LUN string.
[out]	*Lun*	Storage to return the 64-bit LUN.

Return values:

EFI_SUCCESS	The 64-bit LUN is stored in Lun.
EFI_INVALID_PARAMETER	The string is malformatted.

Convert the hexadecimal encoded LUN string into the 64-bit LUN.

Parameters:

[in]	*Str*	The hexadecimal encoded LUN string.
[out]	*Lun*	Storage to return the 64-bit LUN.

Return values:

EFI_SUCCESS	The 64-bit LUN is stored in Lun.
EFI_INVALID_PARAMETER	The string is malformatted.

Referenced by IScsiDhcp6ExtractRootPath(), IScsiDhcpExtractRootPath(), and IScsiFormCallback().

```
EFI_STATUS IScsiBinToHex ( IN UINT8 *          BinBuffer,
                           IN UINT32           BinLength,
                           IN OUT CHAR8 *      HexStr,
                           IN OUT UINT32 *     HexLength
                         )
```

Convert the binary encoded buffer into a hexadecimal encoded string.

Parameters:

[in]	*BinBuffer*	The buffer containing the binary data.
[in]	*BinLength*	Length of the binary buffer.
[in,out]	*HexStr*	Pointer to the string.
[in,out]	*HexLength*	The length of the string.

Return values:

EFI_SUCCESS	The binary data is converted to the hexadecimal string and the length of the string is updated.
EFI_BUFFER_TOO_SMALL	The string is too small.
EFI_INVALID_PARAMETER	The IP string is malformatted.

References IScsiHexString.

Referenced by IScsiCHAPToSendReq().

VOID IScsiCleanDriverData (IN ISCSI_DRIVER_DATA * *Private*)

Clean the iSCSI driver data.

Parameters:

[in] *Private* The iSCSI driver data.

References mPrivate, and ISCSI_PRIVATE_DATA::OneSessionEstablished.

Referenced by IScsiStart(), and IScsiStop().

ISCSI_DRIVER_DATA* IScsiCreateDriverData (IN EFI_HANDLE *Image*, IN EFI_HANDLE *Controller*)

Create the iSCSI driver data.

Parameters:

[in]	*Image*	The handle of the driver image.
[in]	*Controller*	The handle of the controller.

Returns:

The iSCSI driver data created.

Return values:

NULL Other errors as indicated.

References _ISCSI_DRIVER_DATA::Controller, _ISCSI_DRIVER_DATA::ExitBootServiceEvent, _ISCSI_DRIVER_DATA::ExtScsiPassThruHandle, _ISCSI_DRIVER_DATA::ExtScsiPassThruMode, gIScsiExtScsiPassThruProtocolTemplate, _ISCSI_DRIVER_DATA::Image, ISCSI_DRIVER_DATA_SIGNATURE, _ISCSI_DRIVER_DATA::IScsiExtScsiPassThru, IScsiOnExitBootService(), _ISCSI_DRIVER_DATA::Session, and _ISCSI_DRIVER_DATA::Signature.

Referenced by IScsiStart().

```
BOOLEAN IScsiDhcpIsConfigured ( IN EFI_HANDLE  Controller,
                                IN UINT8       IpVersion
                              )
```

Check wheather the Controller handle is configured to use DHCP protocol.

Parameters:

`[in]` *Controller* The handle of the controller.
`[in]` *IpVersion* IP_VERSION_4 or IP_VERSION_6.

Return values:

TRUE The handle of the controller need the Dhcp protocol.
FALSE The handle of the controller does not need the Dhcp protocol.

References gIScsiConfigGuid, IP_MODE_AUTOCONFIG, IP_MODE_IP4, IP_MODE_IP6, ISCSI_DISABLED, ISCSI_MAX_MAC_STRING_LEN, ISCSI_NAME_IFR_MAX_SIZE, IScsiGetVariableAndSize(), and IScsiMacAddrToStr().

Referenced by IScsiSupported().

```
VOID IScsiGenRandom ( IN OUT UINT8 *  Rand,
                      IN UINTN        RandLength
                    )
```

Generate random numbers.

Parameters:

`[in,out]` *Rand* The buffer to contain random numbers.
`[in]` *RandLength* The length of the Rand buffer.

Referenced by IScsiCHAPToSendReq().

```
EFI_STATUS IScsiGetConfigData ( IN ISCSI_DRIVER_DATA *  Private  )
```

Get the various configuration data of this iSCSI instance.

Parameters:

`[in]` *Private* The iSCSI driver data.

Return values:

EFI_SUCCESS Obtained the configuration of this instance.
EFI_ABORTED The operation was aborted.
Others Other errors as indicated.

Get the various configuration data.

Parameters:

`[in]` *Private* The iSCSI driver data.

Return values:

EFI_SUCCESS The configuration data is retrieved.
EFI_NOT_FOUND This iSCSI driver is not configured yet.

References _ISCSI_ATTEMPT_CONFIG_NVDATA::AttemptConfigIndex, ISCSI_PRIVATE_DATA::AttemptConfigs, ISCSI_PRIVATE_DATA::AttemptCount, _ISCSI_ATTEMPT_CONFIG_NVDATA::AutoConfigureMode, _ISCSI_ATTEMPT_CONFIG_NVDATA::AutoConfigureSuccess, ISCSI_NIC_INFO::BusNumber, ISCSI_PRIVATE_DATA::CurrentNic, ISCSI_NIC_INFO::DeviceNumber, _ISCSI_ATTEMPT_CONFIG_NVDATA::DhcpSuccess, _ISCSI_SESSION_CONFIG_NVDATA::Enabled, ISCSI_PRIVATE_DATA::EnableMpio, ISCSI_NIC_INFO::FunctionNumber, gIScsiConfigGuid, gIScsiInitiatorName, ISCSI_NIC_INFO::HwAddressSize, _ISCSI_SESSION_CONFIG_NVDATA::InitiatorInfoFromDhcp, ISCSI_PRIVATE_DATA::InitiatorName, ISCSI_PRIVATE_DATA::InitiatorNameLength, IP_MODE_AUTOCONFIG, IP_MODE_AUTOCONFIG_IP4, IP_MODE_AUTOCONFIG_IP6, IP_MODE_IP4, IP_MODE_IP6, _ISCSI_SESSION_CONFIG_NVDATA::IpMode, IpMode, ISCSI_PRIVATE_DATA::Ipv6Flag, ISCSI_AUTH_TYPE_KRB, ISCSI_CONFIG_VAR_ATTR, ISCSI_DISABLED, ISCSI_ENABLED, ISCSI_ENABLED_FOR_MPIO, ISCSI_MAX_MAC_STRING_LEN, ISCSI_NAME_IFR_MAX_SIZE, ISCSI_NAME_MAX_SIZE, IScsiConfigGetAttemptByConfigIndex(), IScsiConfigUpdateAttempt(), IScsiDoDhcp(), IScsiDoDhcp6(), IScsiGetNicInfoByIndex(), IScsiGetVariableAndSize(), IScsiMacAddrToStr(), ISCSI_PRIVATE_DATA::Krb5MpioCount, _ISCSI_ATTEMPT_CONFIG_NVDATA::Link, _ISCSI_ATTEMPT_CONFIG_NVDATA::MacString, mCallbackInfo, ISCSI_PRIVATE_DATA::MpioCount, mPrivate, ISCSI_NIC_INFO::NicIndex, ISCSI_NIC_INFO::PermanentAddress, ISCSI_PRIVATE_DATA::PortString, _ISCSI_FORM_CALLBACK_INFO::RegisteredHandle, _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData, ISCSI_PRIVATE_DATA::SinglePathCount, _ISCSI_SESSION_CONFIG_NVDATA::TargetInfoFromDhcp, _ISCSI_ATTEMPT_CONFIG_NVDATA::ValidPath, ISCSI_PRIVATE_DATA::ValidSinglePathCount, and ISCSI_NIC_INFO::VlanId.

Referenced by IScsiStart().

ISCSI_NIC_INFO* IScsiGetNicInfoByIndex (IN UINT8 *NicIndex*)

Get the recorded NIC information from a global structure by the Index.

Parameters:

`[in]` *NicIndex* The index indicates the position of NIC info.

Returns:

Pointer to the NIC info or NULL if not found.

Get the recorded NIC info from global structure by the Index.

Parameters:

`[in]` *NicIndex* The index indicates the position of NIC info.

Returns:

Pointer to the NIC info, or NULL if not found.

References mPrivate, ISCSI_NIC_INFO::NicIndex, and ISCSI_PRIVATE_DATA::NicInfoList.

Referenced by IScsiConfigProcessDefault(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiFillNICAndTargetSections(), and IScsiGetConfigData().

UINT16 IScsiGetNICPciLocation (IN EFI_HANDLE *Controller*,

```
                    OUT UINTN *      Bus,
                    OUT UINTN *      Device,
                    OUT UINTN *      Function
                  )
```

Get the NIC's PCI location and return it accroding to the composited format defined in iSCSI Boot Firmware Table.

Parameters:

[in]	*Controller*	The handle of the controller.
[out]	*Bus*	The bus number.
[out]	*Device*	The device number.
[out]	*Function*	The function number.

Returns:

The composited representation of the NIC PCI location.

Referenced by IScsiAddNic().

UINT8 IScsiGetSubnetMaskPrefixLength (IN EFI_IPv4_ADDRESS * *SubnetMask*)

Calculate the prefix length of the IPv4 subnet mask.

Parameters:

[in] *SubnetMask* The IPv4 subnet mask.

Returns:

The prefix length of the subnet mask.

Return values:

0 Other errors as indicated.

Calculate the prefix length of the IPv4 subnet mask.

Parameters:

[in] *SubnetMask* The IPv4 subnet mask.

Returns:

The prefix length of the subnet mask.

Return values:

0 Other errors as indicated.

Referenced by IScsiFillNICAndTargetSections(), and IScsiFormCallback().

EFI_DEVICE_PATH_PROTOCOL* IScsiGetTcpConnDevicePath (IN ISCSI_SESSION * *Session*)

Get the device path of the iSCSI tcp connection and update it.

Parameters:

[in] *Session* The iSCSI session data.

Returns:

The updated device path.

Return values:

NULL Other errors as indicated.

Get the device path of the iSCSI tcp connection and update it.

Parameters:

Session The iSCSI session.

Returns:

The updated device path.

Return values:

NULL Other errors as indicated.

References IP4_NODE_LEN_NEW_VERSIONS, IP6_NODE_LEN_NEW_VERSIONS, IP6_NODE_LEN_OLD_VERSIONS, IP6_OLD_IPADDRESS_OFFSET, _ISCSI_CONNECTION::Ipv6Flag, ISCSI_CONNECTION_SIGNATURE, SESSION_STATE_LOGGED_IN, and _ISCSI_CONNECTION::TcpIo.

Referenced by IScsiStart().

```
VOID* IScsiGetVariableAndSize ( IN CHAR16 *     Name,
                                IN EFI_GUID *   VendorGuid,
                                OUT UINTN *     VariableSize
                              )
```

Read the EFI variable (VendorGuid/Name) and return a dynamically allocated buffer, and the size of the buffer. If failure, return NULL.

Parameters:

[in]	*Name*	String part of EFI variable name.
[in]	*VendorGuid*	GUID part of EFI variable name.
[out]	*VariableSize*	Returns the size of the EFI variable that was read.

Returns:

Dynamically allocated memory that contains a copy of the EFI variable.

Caller is responsible freeing the buffer.

Return values:

NULL Variable was not read.

Referenced by IScsiConfigDeleteAttempts(), IScsiConfigDisplayDeleteAttempts(), IScsiConfigOrderAttempts(), IScsiConfigProcessDefault(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiDhcpIsConfigured(), IScsiGetConfigData(), and IScsiStart().

```
EFI_STATUS IScsiHexToBin ( IN OUT UINT8 *     BinBuffer,
                           IN OUT UINT32 *    BinLength,
                           IN CHAR8 *         HexStr
                         )
```

Convert the hexadecimal string into a binary encoded buffer.

Parameters:

[in,out]	*BinBuffer*	The binary buffer.
[in,out]	*BinLength*	Length of the binary buffer.
[in]	*HexStr*	The hexadecimal string.

Return values:

EFI_SUCCESS	The hexadecimal string is converted into a binary encoded buffer.
EFI_BUFFER_TOO_SMALL	The binary buffer is too small to hold the converted data.

Referenced by IScsiCHAPOnRspReceived().

```
VOID IScsiLunToUnicodeStr ( IN UINT8 *        Lun,
                            OUT CHAR16 *      Str
                          )
```

Convert the 64-bit LUN into the hexadecimal encoded LUN string.

Parameters:

[in]	*Lun*	The 64-bit LUN.
[out]	*String*	The storage to return the hexadecimal encoded LUN string.

Convert the 64-bit LUN into the hexadecimal encoded LUN string.

Parameters:

[in]	*Lun*	The 64-bit LUN.
[out]	*Str*	The storage to return the hexadecimal encoded LUN string.

References IScsiHexString, and IScsiStrTrim().

Referenced by IScsiConvertAttemptConfigDataToIfrNvData().

```
VOID IScsiMacAddrToStr ( IN EFI_MAC_ADDRESS *   Mac,
                         IN UINT32              Len,
                         IN UINT16              VlanId,
                         OUT CHAR16 *           Str
                       )
```

Convert the mac address into a hexadecimal encoded "-" seperated string.

Parameters:

[in]	*Mac*	The mac address.
[in]	*Len*	Length in bytes of the mac address.
[in]	*VlanId*	VLAN ID of the network device.
[out]	*Str*	The storage to return the mac string.

References IScsiHexString.

Referenced by IScsiConfigAddAttempt(), IScsiConfigProcessDefault(), IScsiDhcpIsConfigured(), and IScsiGetConfigData().

```
UINTN IScsiNetNtoi ( IN CHAR8 *   Str   )
```

Convert the decimal-constant string or hex-constant string into a numerical value.

Parameters:

[in] *Str* String in decimal or hex.

Returns:

The numerical value.

Referenced by IScsiCHAPOnRspReceived(), and IScsiCheckOpParams().

VOID EFIAPI IScsiOnExitBootService (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

Abort the session when the transition from BS to RT is initiated.

Parameters:

[in] *Event* The event signaled.
[in] *Context* The iSCSI driver data.

References _ISCSI_DRIVER_DATA::ExitBootServiceEvent, IScsiSessionAbort(), and _ISCSI_DRIVER_DATA::Session.

Referenced by IScsiCreateDriverData().

EFI_STATUS IScsiRemoveNic (IN EFI_HANDLE *Controller*)

Delete the recorded NIC information from a global structure. Also delete corresponding attempts.

Parameters:

[in] *Controller* The handle of the controller.

Return values:

EFI_SUCCESS The operation completed.
EFI_NOT_FOUND The NIC information to be deleted is not recorded.

Delete the recorded NIC info from global structure. Also delete corresponding attempts.

Parameters:

[in] *Controller* The handle of the controller.

Return values:

EFI_SUCCESS The operation is completed.
EFI_NOT_FOUND The NIC info to be deleted is not recorded.

References ISCSI_PRIVATE_DATA::AttemptConfigs, ISCSI_PRIVATE_DATA::AttemptCount, _ISCSI_ATTEMPT_CONFIG_NVDATA::AuthenticationType, ISCSI_PRIVATE_DATA::CurrentNic, _ISCSI_SESSION_CONFIG_NVDATA::Enabled, ISCSI_PRIVATE_DATA::EnableMpio, ISCSI_NIC_INFO::HwAddressSize, ISCSI_AUTH_TYPE_KRB, ISCSI_ENABLED, ISCSI_ENABLED_FOR_MPIO, ISCSI_PRIVATE_DATA::Krb5MpioCount, _ISCSI_ATTEMPT_CONFIG_NVDATA::Link, ISCSI_NIC_INFO::Link, ISCSI_PRIVATE_DATA::MpioCount, mPrivate, ISCSI_PRIVATE_DATA::NewAttempt, ISCSI_PRIVATE_DATA::NicCount, _ISCSI_ATTEMPT_CONFIG_NVDATA::NicIndex,

ISCSI_NIC_INFO::NicIndex, ISCSI_PRIVATE_DATA::NicInfoList, ISCSI_NIC_INFO::PermanentAddress, _ISCSI_ATTEMPT_CONFIG_NVDATA::SessionConfigData, ISCSI_PRIVATE_DATA::SinglePathCount, ISCSI_PRIVATE_DATA::ValidSinglePathCount, and ISCSI_NIC_INFO::VlanId.

Referenced by IScsiStart(), and IScsiStop().

```
EFI_STATUS EFIAPI IScsiTestManagedDevice ( IN EFI_HANDLE  ControllerHandle,
                                           IN EFI_HANDLE  DriverBindingHandle,
                                           IN EFI_GUID *  ProtocolGuid
                                         )
```

Tests whether a controller handle is being managed by IScsi driver.

This function tests whether the driver specified by DriverBindingHandle is currently managing the controller specified by ControllerHandle. This test is performed by evaluating if the the protocol specified by ProtocolGuid is present on ControllerHandle and is was opened by DriverBindingHandle and Nic Device handle with an attribute of EFI_OPEN_PROTOCOL_BY_DRIVER. If ProtocolGuid is NULL, then ASSERT().

Parameters:

ControllerHandle	A handle for a controller to test.
DriverBindingHandle	Specifies the driver binding handle for the driver.
ProtocolGuid	Specifies the protocol that the driver specified by DriverBindingHandle opens in its Start() function.

Return values:

EFI_SUCCESS	ControllerHandle is managed by the driver specified by DriverBindingHandle.
EFI_UNSUPPORTED	ControllerHandle is not managed by the driver specified by DriverBindingHandle.

Referenced by IScsiUnload().

NetworkPkg/IScsiDxe/IScsiProto.c File Reference

Functions

VOID IScsiAttatchConnection (IN OUT ISCSI_SESSION *Session, IN OUT ISCSI_CONNECTION *Conn)
VOID IScsiDetatchConnection (IN OUT ISCSI_CONNECTION *Conn)
EFI_STATUS IScsiCheckSN (IN OUT UINT32 *ExpSN, IN UINT32 NewSN)
VOID IScsiUpdateCmdSN (IN OUT ISCSI_SESSION *Session, IN UINT32 MaxCmdSN, IN UINT32 ExpCmdSN)
EFI_STATUS IScsiConnLogin (IN OUT ISCSI_CONNECTION *Conn, IN UINT16 Timeout)
VOID IScsiConnReset (IN OUT ISCSI_CONNECTION *Conn)
ISCSI_CONNECTION * IScsiCreateConnection (IN ISCSI_SESSION *Session)
VOID IScsiDestroyConnection (IN ISCSI_CONNECTION *Conn)
EFI_STATUS IScsiGetIp6NicInfo (IN ISCSI_CONNECTION *Conn)
EFI_STATUS IScsiSessionLogin (IN ISCSI_SESSION *Session)
EFI_STATUS IScsiSessionReLogin (IN ISCSI_SESSION *Session)
EFI_STATUS IScsiSendLoginReq (IN ISCSI_CONNECTION *Conn)
EFI_STATUS IScsiReceiveLoginRsp (IN ISCSI_CONNECTION *Conn)
EFI_STATUS IScsiAddKeyValuePair (IN OUT NET_BUF *Pdu, IN CHAR8 *Key, IN CHAR8 *Value)
NET_BUF * IScsiPrepareLoginReq (IN OUT ISCSI_CONNECTION *Conn)
EFI_STATUS IScsiProcessLoginRsp (IN OUT ISCSI_CONNECTION *Conn, IN OUT NET_BUF *Pdu)
EFI_STATUS IScsiUpdateTargetAddress (IN OUT ISCSI_SESSION *Session, IN CHAR8 *Data, IN UINT32 Len)
VOID EFIAPI IScsiFreeNbufList (VOID *Arg)
VOID EFIAPI IScsiNbufExtFree (VOID *Arg)
EFI_STATUS IScsiReceivePdu (IN ISCSI_CONNECTION *Conn, OUT NET_BUF **Pdu, IN ISCSI_IN_BUFFER_CONTEXT *Context, IN BOOLEAN HeaderDigest, IN BOOLEAN DataDigest, IN EFI_EVENT TimeoutEvent)
EFI_STATUS IScsiCheckOpParams (IN OUT ISCSI_CONNECTION *Conn)
VOID IScsiFillOpParams (IN ISCSI_CONNECTION *Conn, IN OUT NET_BUF *Pdu)
EFI_STATUS IScsiPadSegment (IN OUT NET_BUF *Pdu, IN UINT32 Len)
LIST_ENTRY * IScsiBuildKeyValueList (IN CHAR8 *Data, IN UINT32 Len)
CHAR8 * IScsiGetValueByKeyFromList (IN OUT LIST_ENTRY *KeyValueList, IN CHAR8 *Key)
VOID IScsiFreeKeyValueList (IN LIST_ENTRY *KeyValueList)
EFI_STATUS IScsiNormalizeName (IN OUT CHAR8 *Name, IN UINTN Len)
EFI_STATUS IScsiNewTcb (IN ISCSI_CONNECTION *Conn, OUT ISCSI_TCB **Tcb)
VOID IScsiDelTcb (IN ISCSI_TCB *Tcb)
ISCSI_TCB * IScsiFindTcbByITT (IN LIST_ENTRY *TcbList, IN UINT32 InitiatorTaskTag)
NET_BUF * IScsiNewDataSegment (IN UINT8 *Data, IN UINT32 Len, IN BOOLEAN DataDigest)
NET_BUF * IScsiNewScsiCmdPdu (IN EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET *Packet, IN UINT64 Lun, IN ISCSI_TCB *Tcb)
NET_BUF * IScsiNewDataOutPdu (IN UINT8 *Data, IN UINT32 Len, IN UINT32 DataSN, IN ISCSI_TCB *Tcb, IN UINT64 Lun)
LIST_ENTRY * IScsiGenerateDataOutPduSequence (IN UINT8 *Data, IN ISCSI_TCB *Tcb, IN UINT64 Lun)
EFI_STATUS

IScsiSendDataOutPduSequence (IN UINT8 *Data, IN UINT64 Lun, IN ISCSI_TCB *Tcb)
EFI_STATUS IScsiOnDataInRcvd (IN NET_BUF *Pdu, IN ISCSI_TCB *Tcb, IN OUT EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET *Packet)
EFI_STATUS IScsiOnR2TRcvd (IN NET_BUF *Pdu, IN ISCSI_TCB *Tcb, IN UINT64 Lun, IN OUT EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET *Packet)
EFI_STATUS IScsiOnScsiRspRcvd (IN NET_BUF *Pdu, IN ISCSI_TCB *Tcb, IN OUT EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET *Packet)
EFI_STATUS IScsiOnNopInRcvd (IN NET_BUF *Pdu, IN ISCSI_TCB *Tcb)
EFI_STATUS IScsiExecuteScsiCommand (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *PassThru, IN UINT8 *Target, IN UINT64 Lun, IN OUT EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET *Packet)
EFI_STATUS IScsiSessionReinstatement (IN ISCSI_SESSION *Session)
VOID IScsiSessionInit (IN OUT ISCSI_SESSION *Session, IN BOOLEAN Recovery)
VOID IScsiSessionAbort (IN OUT ISCSI_SESSION *Session)

Variables

UINT32 mDataSegPad = 0

Detailed Description

The implementation of iSCSI protocol based on RFC3720.

Copyright (c) 2004 - 2014, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS IScsiAddKeyValuePair ( IN OUT NET_BUF *   Pdu,
                                  IN CHAR8 *         Key,
                                  IN CHAR8 *         Value
                                )
```

Add an iSCSI key-value pair as a string into the data segment of the Login Request PDU. The DataSegmentLength and the actual size of the net buffer containing this PDU will be updated.

Parameters:

[in,out]	*Pdu*	The iSCSI PDU whose data segment the key-value pair will be added to.
[in]	*Key*	The key name string.
[in]	*Value*	The value string.

Return values:

EFI_SUCCESS	The key-value pair is added to the PDU's data segment and the correspondence length fields are updated.

EFI_OUT_OF_RESOURCES There is not enough space in the PDU to add the key-value pair.
EFI_PROTOCOL_ERROR There is no such data in the net buffer.

References _ISCSI_LOGIN_REQUEST::DataSegmentLength, ISCSI_SET_DATASEG_LEN, and NTOH24.

Referenced by IScsiCHAPToSendReq(), and IScsiFillOpParams().

```
VOID IScsiAttatchConnection ( IN OUT ISCSI_SESSION *      Session,
                              IN OUT ISCSI_CONNECTION *   Conn
                            )
```

Attach the iSCSI connection to the iSCSI session.

Parameters:

`[in,out]` *Session* The iSCSI session.
`[in,out]` *Conn* The iSCSI connection.

Referenced by IScsiSessionLogin().

```
LIST_ENTRY* IScsiBuildKeyValueList ( IN CHAR8 *   Data,
                                     IN UINT32    Len
                                   )
```

Build a key-value list from the data segment.

Parameters:

`[in]` *Data* The data segment containing the key-value pairs.
`[in]` *Len* Length of the data segment.

Returns:

The key-value list.

Return values:

NULL Other errors as indicated.

References IScsiFreeKeyValueList(), _ISCSI_KEY_VALUE_PAIR::Key, _ISCSI_KEY_VALUE_PAIR::List, and _ISCSI_KEY_VALUE_PAIR::Value.

Referenced by IScsiCHAPOnRspReceived(), IScsiCheckOpParams(), and IScsiUpdateTargetAddress().

```
EFI_STATUS IScsiCheckOpParams ( IN OUT ISCSI_CONNECTION *   Conn )
```

Check and get the result of the parameter negotiation.

Parameters:

`[in,out]` *Conn* The connection in iSCSI login.

Return values:

EFI_SUCCESS The parmeter check is passed and negotiation is finished.
EFI_PROTOCOL_ERROR Some kind of iSCSI protocol error occurred.
EFI_OUT_OF_RESOURCES Failed to allocate memory.

References _ISCSI_SESSION::DataPDUInOrder, _ISCSI_SESSION::DataSequenceInOrder, _ISCSI_SESSION::DefaultTime2Retain, _ISCSI_SESSION::DefaultTime2Wait, _ISCSI_SESSION::ErrorRecoveryLevel, _ISCSI_SESSION::FirstBurstLength, _ISCSI_SESSION::ImmediateData, _ISCSI_SESSION::InitialR2T, ISCSI_KEY_DATA_DIGEST, ISCSI_KEY_DATA_PDU_IN_ORDER, ISCSI_KEY_DATA_SEQUENCE_IN_ORDER, ISCSI_KEY_DEFAULT_TIME2RETAIN, ISCSI_KEY_DEFAULT_TIME2WAIT, ISCSI_KEY_ERROR_RECOVERY_LEVEL, ISCSI_KEY_FIRST_BURST_LENGTH, ISCSI_KEY_HEADER_DIGEST, ISCSI_KEY_IMMEDIATE_DATA, ISCSI_KEY_INITIAL_R2T, ISCSI_KEY_MAX_BURST_LENGTH, ISCSI_KEY_MAX_CONNECTIONS, ISCSI_KEY_MAX_OUTSTANDING_R2T, ISCSI_KEY_MAX_RECV_DATA_SEGMENT_LENGTH, ISCSI_KEY_SESSION_TYPE, ISCSI_KEY_TARGET_ALIAS, ISCSI_KEY_TARGET_PORTAL_GROUP_TAG, ISCSI_KEY_VALUE_NONE, IScsiBuildKeyValueList(), IScsiDigestCRC32, IScsiDigestNone, IScsiFreeKeyValueList(), IScsiGetValueByKeyFromList(), IScsiNetNtoi(), _ISCSI_SESSION::MaxBurstLength, _ISCSI_SESSION::MaxConnections, and _ISCSI_SESSION::MaxOutstandingR2T.

Referenced by IScsiProcessLoginRsp().

```
EFI_STATUS IScsiCheckSN ( IN OUT UINT32 *   ExpSN,
                          IN UINT32         NewSN
                        )
```

Check the sequence number according to RFC3720.

Parameters:

`[in,out]`	*ExpSN*	The currently expected sequence number.
`[in]`	*NewSN*	The sequence number to check.

Return values:

EFI_SUCCESS	The check passed and the ExpSN is increased.
EFI_NOT_READY	Response was sent due to a retransmission request.
EFI_PROTOCOL_ERROR	Some kind of iSCSI protocol error occurred.

References ISCSI_SEQ_EQ, and ISCSI_SEQ_LT.

Referenced by IScsiOnDataInRcvd(), IScsiOnNopInRcvd(), IScsiOnR2TRcvd(), IScsiOnScsiRspRcvd(), and IScsiProcessLoginRsp().

```
EFI_STATUS IScsiConnLogin ( IN OUT ISCSI_CONNECTION *   Conn,
                            IN UINT16                   Timeout
                          )
```

This function does the iSCSI connection login.

Parameters:

`[in,out]`	*Conn*	The iSCSI connection to login.
	Timeout	The timeout value in millisecond.

Return values:

EFI_SUCCESS	The iSCSI connection is logged into the iSCSI target.
EFI_TIMEOUT	Timeout occurred during the login procedure.
Others	Other errors as indicated.

References CONN_STATE_IN_LOGIN, ISCSI_FULL_FEATURE_PHASE, IScsiReceiveLoginRsp(), and IScsiSendLoginReq().

Referenced by IScsiSessionLogin().

VOID IScsiConnReset (IN OUT ISCSI_CONNECTION * *Conn*)

Reset the iSCSI connection.

Parameters:

`[in,out]` *Conn* The iSCSI connection to reset.

Referenced by IScsiSessionAbort(), and IScsiSessionLogin().

ISCSI_CONNECTION* IScsiCreateConnection (IN ISCSI_SESSION * *Session*)

Create a TCP connection for the iSCSI session.

Parameters:

`[in]` *Session* Points to the iSCSI session.

Returns:

The newly created iSCSI connection.

References _ISCSI_CONNECTION::AuthStep, _ISCSI_CONNECTION::Cid, CONN_STATE_FREE, _ISCSI_DRIVER_DATA::Controller, _ISCSI_CONNECTION::CurrentStage, _ISCSI_CONNECTION::DataDigest, DEFAULT_MAX_RECV_DATA_SEG_LEN, _ISCSI_CONNECTION::ExpStatSN, _ISCSI_SESSION_CONFIG_NVDATA::Gateway, _ISCSI_CONNECTION::HeaderDigest, _ISCSI_DRIVER_DATA::Image, IP_MODE_AUTOCONFIG_IP6, IP_MODE_IP6, _ISCSI_SESSION_CONFIG_NVDATA::IpMode, _ISCSI_CONNECTION::Ipv6Flag, ISCSI_AUTH_INITIAL, ISCSI_CONNECTION_SIGNATURE, ISCSI_LOGIN_OPERATIONAL_NEGOTIATION, ISCSI_SECURITY_NEGOTIATION, IScsiDigestNone, _ISCSI_SESSION_CONFIG_NVDATA::LocalIp, _ISCSI_CONNECTION::MaxRecvDataSegmentLength, _ISCSI_CONNECTION::NextStage, _ISCSI_CONNECTION::ParamNegotiated, _ISCSI_CONNECTION::PartialReqSent, _ISCSI_CONNECTION::PartialRspRcvd, _ISCSI_CONNECTION::RspQue, _ISCSI_CONNECTION::Signature, _ISCSI_CONNECTION::State, _ISCSI_SESSION_CONFIG_NVDATA::SubnetMask, _ISCSI_SESSION_CONFIG_NVDATA::TargetIp, _ISCSI_SESSION_CONFIG_NVDATA::TargetPort, _ISCSI_CONNECTION::TcpIo, and _ISCSI_CONNECTION::TimeoutEvent.

Referenced by IScsiSessionLogin().

VOID IScsiDelTcb (IN ISCSI_TCB * *Tcb*)

Delete the tcb from the connection and destroy it.

Parameters:

`[in]` *Tcb* The tcb to delete.

Referenced by IScsiExecuteScsiCommand().

VOID IScsiDestroyConnection (IN ISCSI_CONNECTION *Conn*)

*

Destroy an iSCSI connection.

Parameters:

[in] *Conn* The connection to destroy.

Referenced by IScsiSessionAbort(), and IScsiSessionLogin().

VOID IScsiDetatchConnection (IN OUT ISCSI_CONNECTION * *Conn*)

Detach the iSCSI connection from the session it belongs to.

Parameters:

[in,out] *Conn* The iSCSI connection.

Referenced by IScsiSessionAbort(), and IScsiSessionLogin().

EFI_STATUS IScsiExecuteScsiCommand (IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL * *PassThru*,
IN UINT8 * *Target*,
IN UINT64 *Lun*,
IN OUT EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET * *Packet*
)

Execute the SCSI command issued through the EXT SCSI PASS THRU protocol.

Parameters:

[in]	*PassThru*	The EXT SCSI PASS THRU protocol.
[in]	*Target*	The target ID.
[in]	*Lun*	The LUN.
[in,out]	*Packet*	The request packet containing IO request, SCSI command buffer and buffers to read/write.

Return values:

EFI_SUCCES	The SCSI command is executed and the result is updated to the Packet.
EFI_DEVICE_ERROR	Session state was not as required.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_PROTOCOL_ERROR	There is no such data in the net buffer.
EFI_NOT_READY	The target can not accept new commands.
Others	Other errors as indicated.

References _ISCSI_SESSION::Conns, _ISCSI_XFER_CONTEXT::DesiredLength, _ISCSI_SESSION::FirstBurstLength, _ISCSI_IN_BUFFER_CONTEXT::InData, _ISCSI_IN_BUFFER_CONTEXT::InDataLen, _ISCSI_SESSION::InitialR2T, ISCSI_CONNECTION_SIGNATURE, ISCSI_DRIVER_DATA_FROM_EXT_SCSI_PASS_THRU, ISCSI_GET_DATASEG_LEN, ISCSI_GET_OPCODE, ISCSI_OPCODE_NOP_IN, ISCSI_OPCODE_R2T, ISCSI_OPCODE_SCSI_DATA_IN, ISCSI_OPCODE_SCSI_RSP, ISCSI_OPCODE_VENDOR_T0, ISCSI_OPCODE_VENDOR_T1, ISCSI_OPCODE_VENDOR_T2, ISCSI_RESERVED_TAG, IScsiDelTcb(), IScsiNewScsiCmdPdu(), IScsiNewTcb(), IScsiOnDataInRcvd(), IScsiOnNopInRcvd(),

IScsiOnR2TRcvd(), IScsiOnScsiRspRcvd(), IScsiReceivePdu(), IScsiSendDataOutPduSequence(), _ISCSI_XFER_CONTEXT::Offset, _ISCSI_DRIVER_DATA::Session, SESSION_STATE_LOGGED_IN, _ISCSI_SESSION::State, _ISCSI_TCB::StatusXferd, _ISCSI_XFER_CONTEXT::TargetTransferTag, _ISCSI_CONNECTION::TcpIo, _ISCSI_CONNECTION::TimeoutEvent, and _ISCSI_TCB::XferContext.

Referenced by IScsiExtScsiPassThruFunction().

```
VOID IScsiFillOpParams ( IN ISCSI_CONNECTION *   Conn,
                         IN OUT NET_BUF *        Pdu
                       )
```

Fill the operational parameters.

Parameters:

[in]	*Conn*	The connection in iSCSI login.
[in,out]	*Pdu*	The iSCSI login request PDU to fill the parameters.

References _ISCSI_SESSION::DataPDUInOrder, _ISCSI_SESSION::DataSequenceInOrder, _ISCSI_SESSION::DefaultTime2Retain, _ISCSI_SESSION::DefaultTime2Wait, _ISCSI_SESSION::ErrorRecoveryLevel, _ISCSI_SESSION::FirstBurstLength, _ISCSI_SESSION::ImmediateData, _ISCSI_SESSION::InitialR2T, ISCSI_KEY_DATA_DIGEST, ISCSI_KEY_DATA_PDU_IN_ORDER, ISCSI_KEY_DATA_SEQUENCE_IN_ORDER, ISCSI_KEY_DEFAULT_TIME2RETAIN, ISCSI_KEY_DEFAULT_TIME2WAIT, ISCSI_KEY_ERROR_RECOVERY_LEVEL, ISCSI_KEY_FIRST_BURST_LENGTH, ISCSI_KEY_HEADER_DIGEST, ISCSI_KEY_IMMEDIATE_DATA, ISCSI_KEY_INITIAL_R2T, ISCSI_KEY_MAX_BURST_LENGTH, ISCSI_KEY_MAX_CONNECTIONS, ISCSI_KEY_MAX_OUTSTANDING_R2T, ISCSI_KEY_MAX_RECV_DATA_SEGMENT_LENGTH, IScsiAddKeyValuePair(), IScsiDigestCRC32, MAX_RECV_DATA_SEG_LEN_IN_FFP, _ISCSI_SESSION::MaxBurstLength, _ISCSI_SESSION::MaxConnections, and _ISCSI_SESSION::MaxOutstandingR2T.

Referenced by IScsiPrepareLoginReq().

```
ISCSI_TCB* IScsiFindTcbByITT ( IN LIST_ENTRY *   TcbList,
                               IN UINT32         InitiatorTaskTag
                             )
```

Find the task control block by the initator task tag.

Parameters:

[in]	*TcbList*	The tcb list.
[in]	*InitiatorTaskTag*	The initiator task tag.

Returns:

The task control block found.

Return values:

NULL	The task control block cannot be found.

References _ISCSI_TCB::InitiatorTaskTag.

VOID IScsiFreeKeyValueList (IN LIST_ENTRY * *KeyValueList*)

Free the key-value list.

Parameters:

[in] *KeyValueList* The key-value list.

Referenced by IScsiBuildKeyValueList(), IScsiCHAPOnRspReceived(), IScsiCheckOpParams(), and IScsiUpdateTargetAddress().

VOID EFIAPI IScsiFreeNbufList (VOID * *Arg*)

The callback function to free the net buffer list.

Parameters:

[in] *Arg* The opaque parameter.

Referenced by IScsiGenerateDataOutPduSequence(), IScsiNewDataOutPdu(), IScsiNewScsiCmdPdu(), IScsiReceivePdu(), and IScsiSendDataOutPduSequence().

LIST_ENTRY* IScsiGenerateDataOutPduSequence (IN UINT8 * *Data*,
IN ISCSI_TCB * *Tcb*,
IN UINT64 *Lun*
)

Generate a consecutive sequence of iSCSI SCSI Data Out PDUs.

Parameters:

[in] *Data* The data which will be carried by the sequence of iSCSI SCSI Data Out PDUs.
[in] *Tcb* The task control block of the data to send out.
[in] *Lun* The LUN the data will be sent to.

Returns:

A list of net buffers with each of them wrapping an iSCSI SCSI Data Out PDU.

Return values:

NULL Other errors as indicated.

References _ISCSI_XFER_CONTEXT::DesiredLength, ISCSI_BHS_FLAG_FINAL, ISCSI_SET_FLAG, IScsiFreeNbufList(), IScsiNewDataOutPdu(), _ISCSI_CONNECTION::MaxRecvDataSegmentLength, and _ISCSI_XFER_CONTEXT::Offset.

Referenced by IScsiSendDataOutPduSequence().

EFI_STATUS IScsiGetIp6NicInfo (IN ISCSI_CONNECTION * *Conn*)

Retrieve the IPv6 Address/Prefix/Gateway from the established TCP connection, these informations will be filled in the iSCSI Boot Firmware Table.

Parameters:

[in] *Conn* The connection used in the iSCSI login phase.

Return values:

EFI_SUCCESS Get the NIC information successfully.

Others Other errors as indicated.

References _ISCSI_SESSION_CONFIG_NVDATA::Gateway, _ISCSI_SESSION_CONFIG_NVDATA::LocalIp, _ISCSI_SESSION_CONFIG_NVDATA::PrefixLength, and _ISCSI_SESSION_CONFIG_NVDATA::TargetIp.

Referenced by IScsiSessionLogin().

```
CHAR8* IScsiGetValueByKeyFromList ( IN OUT LIST_ENTRY *   KeyValueList,
                                    IN CHAR8 *            Key
                                  )
```

Get the value string by the key name from the key-value list. If found, the key-value entry will be removed from the list.

Parameters:

`[in,out]` *KeyValueList* The key-value list.
`[in]` *Key* The key name to find.

Returns:

The value string.

Return values:

NULL The key value pair cannot be found.

References _ISCSI_KEY_VALUE_PAIR::Key, _ISCSI_KEY_VALUE_PAIR::List, and _ISCSI_KEY_VALUE_PAIR::Value.

Referenced by IScsiCHAPOnRspReceived(), IScsiCheckOpParams(), and IScsiUpdateTargetAddress().

```
VOID EFIAPI IScsiNbufExtFree ( VOID * Arg )
```

The callback function called in NetBufFree; it does nothing.

Parameters:

`[in]` *Arg* The opaque parameter.

Referenced by IScsiNewDataSegment(), and IScsiReceivePdu().

```
NET_BUF* IScsiNewDataOutPdu ( IN UINT8 *        Data,
                              IN UINT32         Len,
                              IN UINT32         DataSN,
                              IN ISCSI_TCB *    Tcb,
                              IN UINT64         Lun
                            )
```

Create a new iSCSI SCSI Data Out PDU.

Parameters:

`[in]` *Data* The data to put into the Data Out PDU.
`[in]` *Len* Length of the data.
`[in]` *DataSN* The DataSN of the Data Out PDU.

[in] *Tcb* The task control block of this Data Out PDU.
[in] *Lun* The LUN.

Returns:
The net buffer wrapping the Data Out PDU.

Return values:
NULL Other errors as indicated.

References _ISCSI_SCSI_DATA_OUT::BufferOffset, _ISCSI_SCSI_DATA_OUT::DataSN, _ISCSI_SCSI_DATA_OUT::ExpStatSN, _ISCSI_SCSI_DATA_OUT::InitiatorTaskTag, ISCSI_OPCODE_SCSI_DATA_OUT, ISCSI_RESERVED_TAG, ISCSI_SET_DATASEG_LEN, ISCSI_SET_OPCODE, IScsiFreeNbufList(), IScsiNewDataSegment(), _ISCSI_SCSI_DATA_OUT::Lun, _ISCSI_XFER_CONTEXT::Offset, _ISCSI_XFER_CONTEXT::TargetTransferTag, and _ISCSI_SCSI_DATA_OUT::TargetTransferTag.

Referenced by IScsiGenerateDataOutPduSequence().

```
NET_BUF* IScsiNewDataSegment ( IN UINT8 *      Data,
                               IN UINT32       Len,
                               IN BOOLEAN      DataDigest
                             )
```

Create a data segment, pad it, and calculate the CRC if needed.

Parameters:
[in] *Data* The data to fill into the data segment.
[in] *Len* Length of the data.
[in] *DataDigest* Whether to calculate CRC for this data segment.

Returns:
The net buffer wrapping the data segment.

References ISCSI_GET_PAD_LEN, IScsiNbufExtFree(), and mDataSegPad.

Referenced by IScsiNewDataOutPdu(), and IScsiNewScsiCmdPdu().

```
NET_BUF*                    ( IN EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET *  Packet,
IScsiNewScsiCmdPdu            IN UINT64                                        Lun,
                              IN ISCSI_TCB *                                   Tcb
                            )
```

Create a iSCSI SCSI command PDU to encapsulate the command issued by SCSI through the EXT SCSI PASS THRU Protocol.

Parameters:
[in] *Packet* The EXT SCSI PASS THRU request packet containing the SCSI command.
[in] *Lun* The LUN.
[in] *Tcb* The tcb assocated with this SCSI command.

Returns:

The created iSCSI SCSI command PDU.

Return values:

NULL Other errors as indicated.

References _SCSI_COMMAND::Cdb, _SCSI_COMMAND::CmdSN, DataBi, DataIn, DataOut, _SCSI_COMMAND::ExpDataXferLength, _ISCSI_BI_EXP_READ_DATA_LEN_AHS::ExpReadDataLength, _SCSI_COMMAND::ExpStatSN, _ISCSI_SESSION::FirstBurstLength, _ISCSI_SESSION::ImmediateData, _ISCSI_SESSION::InitialR2T, _SCSI_COMMAND::InitiatorTaskTag, ISCSI_AHS_TYPE_BI_EXP_READ_DATA_LEN, ISCSI_AHS_TYPE_EXT_CDB, ISCSI_BHS_FLAG_FINAL, ISCSI_OPCODE_SCSI_CMD, ISCSI_ROUNDUP, ISCSI_SET_DATASEG_LEN, ISCSI_SET_FLAG, ISCSI_SET_OPCODE, ISCSI_TASK_ATTR_SIMPLE, IScsiFreeNbufList(), IScsiNewDataSegment(), _ISCSI_ADDTIONAL_HEADER::Length, _ISCSI_BI_EXP_READ_DATA_LEN_AHS::Length, _SCSI_COMMAND::Lun, SCSI_CMD_PDU_FLAG_READ, SCSI_CMD_PDU_FLAG_WRITE, _SCSI_COMMAND::TotalAHSLength, _ISCSI_ADDTIONAL_HEADER::Type, and _ISCSI_BI_EXP_READ_DATA_LEN_AHS::Type.

Referenced by IScsiExecuteScsiCommand().

```
EFI_STATUS IScsiNewTcb ( IN ISCSI_CONNECTION *   Conn,
                         OUT ISCSI_TCB **        Tcb
                       )
```

Create an iSCSI task control block.

Parameters:

[in]	*Conn*	The connection on which the task control block will be created.
[out]	*Tcb*	The newly created task control block.

Return values:

EFI_SUCCESS	The task control block is created.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_NOT_READY	The target cannot accept new commands.

References _ISCSI_TCB::CmdSN, _ISCSI_SESSION::CmdSN, _ISCSI_TCB::Conn, _ISCSI_SESSION::InitiatorTaskTag, _ISCSI_TCB::InitiatorTaskTag, ISCSI_SEQ_GT, _ISCSI_TCB::Link, _ISCSI_SESSION::MaxCmdSN, _ISCSI_TCB::SoFarInOrder, and _ISCSI_SESSION::TcbList.

Referenced by IScsiExecuteScsiCommand().

```
EFI_STATUS IScsiNormalizeName ( IN OUT CHAR8 *   Name,
                                IN UINTN         Len
                              )
```

Normalize the iSCSI name according to RFC.

Parameters:

[in,out]	*Name*	The iSCSI name.
[in]	*Len*	Length of the iSCSI name.

Return values:

EFI_SUCCESS	The iSCSI name is valid and normalized.

EFI_PROTOCOL_ERROR The iSCSI name is malformatted or not in the IQN format.

Referenced by IScsiDhcp6ExtractRootPath(), IScsiDhcpExtractRootPath(), IScsiFormCallback(), and IScsiSetInitiatorName().

```
EFI_STATUS
IScsiOnDataInRcvd    (  IN NET_BUF *                                  Pdu,
                        IN ISCSI_TCB *                                Tcb,
                        IN OUT
                        EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET    Packet
                        *
                     )
```

Process the received iSCSI SCSI Data In PDU.

Parameters:

[in]	*Pdu*	The Data In PDU received.
[in]	*Tcb*	The task control block.
[in,out]	*Packet*	The EXT SCSI PASS THRU request packet.

Return values:

EFI_SUCCES	The check on the Data IN PDU is passed and some update actions are taken.
EFI_PROTOCOL_ERROR	Some kind of iSCSI protocol errror occurred.
EFI_BAD_BUFFER_SIZEE	The buffer was not the proper size for the request.
Others	Other errors as indicated.

References _ISCSI_SCSI_DATA_IN::DataSN, _ISCSI_SCSI_DATA_IN::ExpCmdSN, _ISCSI_SCSI_DATA_IN::InitiatorTaskTag, ISCSI_BHS_FLAG_FINAL, ISCSI_FLAG_ON, IScsiCheckSN(), IScsiUpdateCmdSN(), _ISCSI_SCSI_DATA_IN::MaxCmdSN, _ISCSI_SCSI_DATA_IN::ResidualCount, SCSI_DATA_IN_PDU_FLAG_OVERFLOW, SCSI_DATA_IN_PDU_FLAG_STATUS_VALID, SCSI_DATA_IN_PDU_FLAG_UNDERFLOW, SCSI_RSP_PDU_FLAG_OVERFLOW, SCSI_RSP_PDU_FLAG_UNDERFLOW, _ISCSI_SCSI_DATA_IN::StatSN, and _ISCSI_SCSI_DATA_IN::Status.

Referenced by IScsiExecuteScsiCommand().

```
EFI_STATUS IScsiOnNopInRcvd    (  IN
                                  NET_BUF *    Pdu,
                                  IN
                                  ISCSI_TCB    Tcb
                                  *
                               )
```

Process the received NOP In PDU.

Parameters:

[in]	*Pdu*	The NOP In PDU received.
[in]	*Tcb*	The task control block.

Return values:

EFI_SUCCES

	The NOP In PDU is processed and the related sequence numbers are updated.
EFI_PROTOCOL_ERROR	Some kind of iSCSI protocol errror occurred.

References _ISCSI_NOP_IN::ExpCmdSN, _ISCSI_NOP_IN::InitiatorTaskTag, ISCSI_RESERVED_TAG, IScsiCheckSN(), IScsiUpdateCmdSN(), _ISCSI_NOP_IN::MaxCmdSN, and _ISCSI_NOP_IN::StatSN.

Referenced by IScsiExecuteScsiCommand().

```
EFI_STATUS
IScsiOnR2TRcvd          ( IN NET_BUF *                                    Pdu,
                          IN ISCSI_TCB *                                  Tcb,
                          IN UINT64                                       Lun,
                          IN OUT
                          EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET *    Packet
                        )
```

Process the received iSCSI R2T PDU.

Parameters:

`[in]`	*Pdu*	The R2T PDU received.
`[in]`	*Tcb*	The task control block.
`[in]`	*Lun*	The Lun.
`[in,out]`	*Packet*	The EXT SCSI PASS THRU request packet.

Return values:

EFI_SUCCES	The R2T PDU is valid and the solicited data is sent out.
EFI_PROTOCOL_ERROR	Some kind of iSCSI protocol errror occurred.
Others	Other errors as indicated.

References _ISCSI_READY_TO_TRANSFER::BufferOffset, _ISCSI_READY_TO_TRANSFER::DesiredDataTransferLength, _ISCSI_XFER_CONTEXT::DesiredLength, _ISCSI_READY_TO_TRANSFER::InitiatorTaskTag, ISCSI_SEQ_EQ, IScsiCheckSN(), IScsiSendDataOutPduSequence(), _ISCSI_XFER_CONTEXT::Offset, _ISCSI_READY_TO_TRANSFER::R2TSeqNum, _ISCSI_READY_TO_TRANSFER::StatSN, _ISCSI_XFER_CONTEXT::TargetTransferTag, and _ISCSI_READY_TO_TRANSFER::TargetTransferTag.

Referenced by IScsiExecuteScsiCommand().

```
EFI_STATUS
IScsiOnScsiRspRcvd      ( IN NET_BUF *                                    Pdu,
                          IN ISCSI_TCB *                                  Tcb,
                          IN OUT
                          EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET *    Packet
                        )
```

Process the received iSCSI SCSI Response PDU.

Parameters:

`[in]`	*Pdu*	The Response PDU received.
`[in]`	*Tcb*	The task control block.
`[in,out]`	*Packet*	The EXT SCSI PASS THRU request packet.

Return values:

EFI_SUCCES	The Response PDU is processed.
EFI_PROTOCOL_ERROR	Some kind of iSCSI protocol errror occurred.
EFI_BAD_BUFFER_SIZEE	The buffer was not the proper size for the request.
Others	Other errors as indicated.

References _SCSI_RESPONSE::BiReadResidualCount, _ISCSI_SENSE_DATA::Data, DataIn, _SCSI_RESPONSE::ExpCmdSN, _SCSI_RESPONSE::InitiatorTaskTag, ISCSI_FLAG_ON, ISCSI_GET_DATASEG_LEN, ISCSI_SERVICE_RSP_COMMAND_COMPLETE_AT_TARGET, IScsiCheckSN(), IScsiUpdateCmdSN(), _ISCSI_SENSE_DATA::Length, _SCSI_RESPONSE::MaxCmdSN, _SCSI_RESPONSE::ResidualCount, _SCSI_RESPONSE::Response, SCSI_RSP_PDU_FLAG_BI_READ_OVERFLOW, SCSI_RSP_PDU_FLAG_BI_READ_UNDERFLOW, SCSI_RSP_PDU_FLAG_OVERFLOW, SCSI_RSP_PDU_FLAG_UNDERFLOW, _SCSI_RESPONSE::StatSN, and _SCSI_RESPONSE::Status.

Referenced by IScsiExecuteScsiCommand().

```
EFI_STATUS IScsiPadSegment ( IN OUT NET_BUF *   Pdu,
                             IN UINT32          Len
                           )
```

Pad the iSCSI AHS or data segment to an integer number of 4 byte words.

Parameters:

`[in,out]`	*Pdu*	The iSCSI pdu which contains segments to pad.
`[in]`	*Len*	The length of the last segment in the PDU.

Return values:

EFI_SUCCESS	The segment is padded or there is no need to pad it.
EFI_OUT_OF_RESOURCES	There is not enough remaining free space to add the padding bytes.

References ISCSI_GET_PAD_LEN.

Referenced by IScsiPrepareLoginReq().

```
NET_BUF* IScsiPrepareLoginReq ( IN OUT ISCSI_CONNECTION *   Conn   )
```

Prepare the iSCSI login request to be sent according to the current login status.

Parameters:

`[in,out]`	*Conn*	The connection in the iSCSI login phase.

Returns:

The pointer to the net buffer containing the iSCSI login request built.

Return values:

NULL	Other errors as indicated.

References _ISCSI_SESSION::AuthType, _ISCSI_LOGIN_REQUEST::Cid, _ISCSI_SESSION::CmdSN, _ISCSI_LOGIN_REQUEST::CmdSN, DEFAULT_MAX_RECV_DATA_SEG_LEN, _ISCSI_LOGIN_REQUEST::ExpStatSN, _ISCSI_SESSION::InitiatorTaskTag, _ISCSI_LOGIN_REQUEST::InitiatorTaskTag, ISCSI_AUTH_TYPE_KRB, ISCSI_FLAG_ON, ISCSI_GET_DATASEG_LEN, ISCSI_LOGIN_OPERATIONAL_NEGOTIATION,

ISCSI_LOGIN_REQ_PDU_FLAG_TRANSIT, ISCSI_OPCODE_LOGIN_REQ, ISCSI_REQ_IMMEDIATE, ISCSI_SECURITY_NEGOTIATION, ISCSI_SET_FLAG, ISCSI_SET_OPCODE, ISCSI_SET_STAGES, ISCSI_VERSION_MAX, ISCSI_VERSION_MIN, IScsiCHAPToSendReq(), IScsiFillOpParams(), IScsiPadSegment(), _ISCSI_SESSION::Isid, _ISCSI_LOGIN_REQUEST::Isid, _ISCSI_SESSION::Tsih, _ISCSI_LOGIN_REQUEST::Tsih, _ISCSI_LOGIN_REQUEST::VersionMax, and _ISCSI_LOGIN_REQUEST::VersionMin.

Referenced by IScsiSendLoginReq().

```
EFI_STATUS IScsiProcessLoginRsp ( IN OUT ISCSI_CONNECTION *   Conn,
                                  IN OUT NET_BUF *            Pdu
                                )
```

Process the iSCSI Login Response.

Parameters:

[in,out]	*Conn*	The connection on which the iSCSI login response is received.
[in,out]	*Pdu*	The iSCSI login response PDU.

Return values:

EFI_SUCCESS	The iSCSI login response PDU is processed, and all checks are passed.
EFI_PROTOCOL_ERROR	Some kind of iSCSI protocol error occurred.
EFI_MEDIA_CHANGED	Target is redirected.
Others	Other errors as indicated.

References _ISCSI_SESSION::AuthType, _ISCSI_SESSION::CmdSN, _ISCSI_SESSION::ExpCmdSN, _ISCSI_LOGIN_RESPONSE::ExpCmdSN, _ISCSI_SESSION::InitiatorTaskTag, _ISCSI_LOGIN_RESPONSE::InitiatorTaskTag, ISCSI_AUTH_INITIAL, ISCSI_AUTH_TYPE_KRB, ISCSI_CHECK_OPCODE, ISCSI_FLAG_ON, ISCSI_FULL_FEATURE_PHASE, ISCSI_GET_CURRENT_STAGE, ISCSI_GET_DATASEG_LEN, ISCSI_GET_NEXT_STAGE, ISCSI_LOGIN_OPERATIONAL_NEGOTIATION, ISCSI_LOGIN_RSP_PDU_FLAG_CONTINUE, ISCSI_LOGIN_RSP_PDU_FLAG_TRANSIT, ISCSI_LOGIN_STATUS_REDIRECTION, ISCSI_LOGIN_STATUS_SUCCESS, ISCSI_OPCODE_LOGIN_RSP, ISCSI_SECURITY_NEGOTIATION, IScsiCHAPOnRspReceived(), IScsiCheckOpParams(), IScsiCheckSN(), IScsiUpdateCmdSN(), IScsiUpdateTargetAddress(), _ISCSI_LOGIN_RESPONSE::Isid, _ISCSI_SESSION::Isid, _ISCSI_SESSION::MaxCmdSN, _ISCSI_LOGIN_RESPONSE::MaxCmdSN, SESSION_STATE_FREE, _ISCSI_SESSION::State, _ISCSI_LOGIN_RESPONSE::StatSN, _ISCSI_LOGIN_RESPONSE::StatusClass, _ISCSI_LOGIN_RESPONSE::Tsih, and _ISCSI_SESSION::Tsih.

Referenced by IScsiReceiveLoginRsp().

```
EFI_STATUS IScsiReceiveLoginRsp ( IN ISCSI_CONNECTION *   Conn )
```

Receive and process the iSCSI login response.

Parameters:

[in]	*Conn*	The connection in the iSCSI login phase.

Return values:

EFI_SUCCESS	The iSCSI login response PDU is received and processed.

Others Other errors as indicated.

References IScsiProcessLoginRsp(), and IScsiReceivePdu().

Referenced by IScsiConnLogin().

```
EFI_STATUS IScsiReceivePdu ( IN ISCSI_CONNECTION *          Conn,
                             OUT NET_BUF **                 Pdu,
                             IN ISCSI_IN_BUFFER_CONTEXT *   Context,
                             IN BOOLEAN                     HeaderDigest,
                             IN BOOLEAN                     DataDigest,
                             IN EFI_EVENT                   TimeoutEvent
                           )
```

Receive an iSCSI response PDU. An iSCSI response PDU contains an iSCSI PDU header and an optional data segment. The two parts will be put into two blocks of buffers in the net buffer. The digest check will be conducted in this function if needed and the digests will be trimmed from the PDU buffer.

Parameters:

[in]	*Conn*	The iSCSI connection to receive data from.
[out]	*Pdu*	The received iSCSI pdu.
[in]	*Context*	The context used to describe information on the caller provided buffer to receive data segment of the iSCSI pdu. It is optional.
[in]	*HeaderDigest*	Whether there will be header digest received.
[in]	*DataDigest*	Whether there will be data digest.
[in]	*TimeoutEvent*	The timeout event. It is optional.

Return values:

EFI_SUCCESS	An iSCSI pdu is received.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_PROTOCOL_ERROR	Some kind of iSCSI protocol error occurred.
Others	Other errors as indicated.

References ISCSI_GET_BUFFER_OFFSET, ISCSI_GET_DATASEG_LEN, ISCSI_GET_OPCODE, ISCSI_GET_PAD_LEN, ISCSI_OPCODE_ASYNC_MSG, ISCSI_OPCODE_LOGIN_RSP, ISCSI_OPCODE_NOP_IN, ISCSI_OPCODE_REJECT, ISCSI_OPCODE_SCSI_DATA_IN, ISCSI_OPCODE_SCSI_RSP, ISCSI_OPCODE_TEXT_RSP, ISCSI_OPCODE_VENDOR_T0, ISCSI_OPCODE_VENDOR_T1, ISCSI_OPCODE_VENDOR_T2, IScsiFreeNbufList(), and IScsiNbufExtFree().

Referenced by IScsiExecuteScsiCommand(), and IScsiReceiveLoginRsp().

```
EFI_STATUS IScsiSendDataOutPduSequence ( IN UINT8 *        Data,
                                         IN UINT64         Lun,
                                         IN ISCSI_TCB *    Tcb
                                       )
```

Send the Data in a sequence of Data Out PDUs one by one.

Parameters:

[in] *Data* The data to carry by Data Out PDUs.

[in] *Lun* The LUN the data will be sent to.
[in] *Tcb* The task control block.

Return values:

EFI_SUCCES	The data is sent out to the LUN.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
Others	Other errors as indicated.

References IScsiFreeNbufList(), and IScsiGenerateDataOutPduSequence().

Referenced by IScsiExecuteScsiCommand(), and IScsiOnR2TRcvd().

EFI_STATUS IScsiSendLoginReq (IN ISCSI_CONNECTION * *Conn*)

Build and send the iSCSI login request to the iSCSI target according to the current login stage.

Parameters:

[in] *Conn* The connection in the iSCSI login phase.

Return values:

EFI_SUCCESS	The iSCSI login request PDU is built and sent on this connection.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_DEVICE_ERROR	Some kind of device error occurred.

References IScsiPrepareLoginReq().

Referenced by IScsiConnLogin().

VOID IScsiSessionAbort (IN OUT ISCSI_SESSION * *Session*)

Abort the iSCSI session. That is, reset all the connection(s), and free the resources.

Parameters:

[in,out] *Session* The iSCSI session.

References _ISCSI_CONNECTION::Ipv6Flag, ISCSI_CONNECTION_SIGNATURE, IScsiConnReset(), IScsiDestroyConnection(), IScsiDetatchConnection(), SESSION_STATE_FAILED, SESSION_STATE_LOGGED_IN, and _ISCSI_CONNECTION::TcpIo.

Referenced by IScsiOnExitBootService(), IScsiSessionReinstatement(), IScsiStart(), and IScsiStop().

VOID IScsiSessionInit (IN OUT ISCSI_SESSION * *Session*,
IN BOOLEAN *Recovery*
)

Initialize some session parameters before login.

Parameters:

[in,out] *Session* The iSCSI session.
[in] *Recovery* Whether the request is from a fresh new start or recovery.

References DEFAULT_MAX_OUTSTANDING_R2T, ISCSI_MAX_CONNS_PER_SESSION, ISCSI_SESSION_SIGNATURE, MAX_RECV_DATA_SEG_LEN_IN_FFP, and SESSION_STATE_FREE.

Referenced by IScsiSessionReinstatement(), and IScsiStart().

EFI_STATUS IScsiSessionLogin (IN ISCSI_SESSION * *Session*)
Login the iSCSI session.

Parameters:

[in] *Session* The iSCSI session.

Return values:

EFI_SUCCESS	The iSCSI session login procedure finished.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_NO_MEDIA	There was a media error.
Others	Other errors as indicated.

References _ISCSI_CONNECTION::Ipv6Flag, IScsiAttatchConnection(), IScsiConnLogin(), IScsiConnReset(), IScsiCreateConnection(), IScsiDestroyConnection(), IScsiDetatchConnection(), IScsiGetIp6NicInfo(), SESSION_STATE_LOGGED_IN, and _ISCSI_CONNECTION::TcpIo.

Referenced by IScsiSessionReinstatement(), IScsiSessionReLogin(), and IScsiStart().

EFI_STATUS IScsiSessionReinstatement (IN ISCSI_SESSION * *Session*)
Reinstate the session on some error.

Parameters:

[in] *Session* The iSCSI session

Return values:

EFI_SUCCESS	The session is reinstated from some error.
Other	Reinstatement failed.

References IScsiSessionAbort(), IScsiSessionInit(), IScsiSessionLogin(), and SESSION_STATE_FREE.

Referenced by IScsiExtScsiPassThruFunction().

EFI_STATUS IScsiSessionReLogin (IN ISCSI_SESSION * *Session*)
Wait for IPsec negotiation, then try to login the iSCSI session again.

Parameters:

[in] *Session* The iSCSI session.

Return values:

EFI_SUCCESS	The iSCSI session login procedure finished.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_PROTOCOL_ERROR	Some kind of iSCSI protocol error occurred.

References ISCSI_WAIT_IPSEC_TIMEOUT, and IScsiSessionLogin().

Referenced by IScsiStart().

```
VOID IScsiUpdateCmdSN ( IN OUT ISCSI_SESSION *   Session,
                        IN UINT32                MaxCmdSN,
                        IN UINT32                ExpCmdSN
                      )
```

Update the sequence numbers for the iSCSI command.

Parameters:

[in,out]	*Session*	The iSCSI session.
[in]	*MaxCmdSN*	Maximum CmdSN from the target.
[in]	*ExpCmdSN*	Next expected CmdSN from the target.

References ISCSI_SEQ_GT, and ISCSI_SEQ_LT.

Referenced by IScsiOnDataInRcvd(), IScsiOnNopInRcvd(), IScsiOnScsiRspRcvd(), and IScsiProcessLoginRsp().

```
EFI_STATUS IScsiUpdateTargetAddress ( IN OUT ISCSI_SESSION *   Session,
                                      IN CHAR8 *               Data,
                                      IN UINT32                Len
                                    )
```

Updated the target information according the data received in the iSCSI login response with an target redirection status.

Parameters:

[in,out]	*Session*	The iSCSI session.
[in]	*Data*	The data segment that should contain the TargetAddress key-value list.
[in]	*Len*	Length of the data.

Return values:

EFI_SUCCESS	The target address is updated.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_NOT_FOUND	The TargetAddress key is not found.
Others	Other errors as indicated.

References IP_MODE_AUTOCONFIG, IpMode, ISCSI_KEY_TARGET_ADDRESS, ISCSI_WELL_KNOWN_PORT, IScsiAsciiStrToIp(), IScsiBuildKeyValueList(), IScsiFreeKeyValueList(), and IScsiGetValueByKeyFromList().

Referenced by IScsiProcessLoginRsp().

Variable Documentation

UINT32 mDataSegPad = 0

Referenced by IScsiNewDataSegment().

- Data Structures
- Files

- File List
- Globals

NetworkPkg/IScsiDxe/IScsiProto.h File Reference

Data Structures

struct _ISCSI_BASIC_HEADER
struct _ISCSI_ADDTIONAL_HEADER
struct _ISCSI_BI_EXP_READ_DATA_LEN_A
struct _SCSI_COMMAND
struct _SCSI_RESPONSE
struct _ISCSI_SENSE_DATA
struct _ISCSI_TMF_REQUEST
struct _ISCSI_TMF_RESPONSE
struct _ISCSI_SCSI_DATA_OUT
struct _ISCSI_SCSI_DATA_IN
struct _ISCSI_READY_TO_TRANSFER
struct _ISCSI_ASYNC_MESSAGE
struct _ISCSI_LOGIN_REQUEST
struct _ISCSI_LOGIN_RESPONSE
struct _ISCSI_LOGOUT_REQUEST
struct _ISCSI_LOGOUT_RESPONSE
struct _ISCSI_SNACK_REQUEST
struct _ISCSI_REJECT
struct _ISCSI_NOP_OUT
struct _ISCSI_NOP_IN
struct _ISCSI_XFER_CONTEXT
struct _ISCSI_IN_BUFFER_CONTEXT
struct _ISCSI_TCB
struct _ISCSI_KEY_VALUE_PAIR

Defines

"memItemRight" valign="bottom">ISCSI_SEQ_EQ(s1, s2) ((s1) == (s2))
#define ISCSI_SEQ_LT(s1, s2)
#define ISCSI_SEQ_GT(s1, s2)
#define ISCSI_WELL_KNOWN_PORT 3260
#define ISCSI_MAX_CONNS_PER_SESSION
#define DEFAULT_MAX_RECV_DATA_SEG_
#define MAX_RECV_DATA_SEG_LEN_IN_FF
#define DEFAULT_MAX_OUTSTANDING_R2
#define ISCSI_VERSION_MAX 0x00
#define ISCSI_VERSION_MIN 0x00
#define ISCSI_KEY_AUTH_METHOD "AuthM
#define ISCSI_KEY_HEADER_DIGEST "Head
#define ISCSI_KEY_DATA_DIGEST "DataDig
#define ISCSI_KEY_MAX_CONNECTIONS "
#define ISCSI_KEY_TARGET_NAME "Target
#define ISCSI_KEY_INITIATOR_NAME "Initi
#define ISCSI_KEY_TARGET_ALIAS "Target
#define ISCSI_KEY_INITIATOR_ALIAS "Initi
#define ISCSI_KEY_TARGET_ADDRESS "Ta
#define ISCSI_KEY_INITIAL_R2T "InitialR2T
#define ISCSI_KEY_IMMEDIATE_DATA "Im

```
#define ISCSI_KEY_TARGET_PORTAL_GROUP_TAG
#define ISCSI_KEY_MAX_BURST_LENGTH "MaxBu
#define ISCSI_KEY_FIRST_BURST_LENGTH "FirstB
#define ISCSI_KEY_DEFAULT_TIME2WAIT "Defaul
#define ISCSI_KEY_DEFAULT_TIME2RETAIN "Defa
#define ISCSI_KEY_MAX_OUTSTANDING_R2T "Ma
#define ISCSI_KEY_DATA_PDU_IN_ORDER "DataPI
#define ISCSI_KEY_DATA_SEQUENCE_IN_ORDER
#define ISCSI_KEY_ERROR_RECOVERY_LEVEL "E
#define ISCSI_KEY_SESSION_TYPE "SessionType"
#define ISCSI_KEY_MAX_RECV_DATA_SEGMENT_
#define ISCSI_KEY_VALUE_NONE "None"
#define CONN_STATE_FREE 0
#define CONN_STATE_XPT_WAIT 1
#define CONN_STATE_IN_LOGIN 2
#define CONN_STATE_LOGGED_IN 3
#define CONN_STATE_IN_LOGOUT 4
#define CONN_STATE_LOGOUT_REQUESTED 5
#define CONN_STATE_CLEANUP_WAIT 6
#define CONN_STATE_IN_CLEANUP 7
#define SESSION_STATE_FREE 0
#define SESSION_STATE_LOGGED_IN 1
#define SESSION_STATE_FAILED 2
#define ISCSI_RESERVED_TAG 0xffffffff
#define ISCSI_REQ_IMMEDIATE 0x40
#define ISCSI_OPCODE_MASK 0x3F
#define ISCSI_SET_OPCODE(PduHdr, Op, Flgs) ((((IS
        = ((Op) | (Flgs)))
#define ISCSI_GET_OPCODE(PduHdr) ((((ISCSI_BAS
        ISCSI_OPCODE_MASK)
#define ISCSI_CHECK_OPCODE(PduHdr, Op) ((((Pdu
        (Op))
#define ISCSI_IMMEDIATE_ON(PduHdr) ((PduHdr)->
#define ISCSI_SET_FLAG(PduHdr, Flag) (((ISCSI_BA
        (BOOLEAN)(Flag))
#define ISCSI_CLEAR_FLAG(PduHdr, Flag) (((ISCSI_
        ~(Flag))
#define ISCSI_FLAG_ON(PduHdr, Flag) ((BOOLEAN)
        (PduHdr))->Flags & (Flag)) == (Flag)))
#define ISCSI_SET_STAGES(PduHdr, Cur, Nxt) ((PduH
        << 2 | (Nxt))))
#define ISCSI_GET_CURRENT_STAGE(PduHdr) ((UI
#define ISCSI_GET_NEXT_STAGE(PduHdr) ((UINT8)
#define ISCSI_GET_PAD_LEN(DataLen) ((~(DataLen)
#define ISCSI_ROUNDUP(DataLen) (((DataLen) + 3) &
#define HTON24(Dst, Src)
#define NTOH24(src) (((src)[0] << 16) | ((src)[1] << 8) |
#define ISCSI_GET_DATASEG_LEN(PduHdr) NTOH2
        (PduHdr))->DataSegmentLength)
#define ISCSI_SET_DATASEG_LEN(PduHdr, Len) HT
        (PduHdr))->DataSegmentLength, (Len))
#define ISCSI_GET_BUFFER_OFFSET(PduHdr) NTOH
        (PduHdr))->BufferOffset)
#define ISCSI_OPCODE_NOP_OUT 0x00
```

```
#define ISCSI_OPCODE_SCSI_CMD 0x01
#define ISCSI_OPCODE_SCSI_TMF_REQ 0x0
#define ISCSI_OPCODE_LOGIN_REQ 0x03
#define ISCSI_OPCODE_TEXT_REQ 0x04
#define ISCSI_OPCODE_SCSI_DATA_OUT 0
#define ISCSI_OPCODE_LOGOUT_REQ 0x06
#define ISCSI_OPCODE_SNACK_REQ 0x10
#define ISCSI_OPCODE_VENDOR_I0 0x1c
#define ISCSI_OPCODE_VENDOR_I1 0x1d
#define ISCSI_OPCODE_VENDOR_I2 0x1e
#define ISCSI_OPCODE_NOP_IN 0x20
#define ISCSI_OPCODE_SCSI_RSP 0x21
#define ISCSI_OPCODE_SCSI_TMF_RSP 0x2
#define ISCSI_OPCODE_LOGIN_RSP 0x23
#define ISCSI_OPCODE_TEXT_RSP 0x24
#define ISCSI_OPCODE_SCSI_DATA_IN 0x2
#define ISCSI_OPCODE_LOGOUT_RSP 0x26
#define ISCSI_OPCODE_R2T 0x31
#define ISCSI_OPCODE_ASYNC_MSG 0x32
#define ISCSI_OPCODE_VENDOR_T0 0x3c
#define ISCSI_OPCODE_VENDOR_T1 0x3d
#define ISCSI_OPCODE_VENDOR_T2 0x3e
#define ISCSI_OPCODE_REJECT 0x3f
#define ISCSI_BHS_FLAG_FINAL 0x80
#define ISCSI_AHS_TYPE_EXT_CDB 0x1
#define ISCSI_AHS_TYPE_BI_EXP_READ_DA
#define SCSI_CMD_PDU_FLAG_READ 0x40
#define SCSI_CMD_PDU_FLAG_WRITE 0x20
#define ISCSI_CMD_PDU_TASK_ATTR_MAS
#define ISCSI_TASK_ATTR_UNTAGGED 0x0
#define ISCSI_TASK_ATTR_SIMPLE 0x01
#define ISCSI_TASK_ATTR_ORDERD 0x02
#define ISCSI_TASK_ATTR_HOQ 0x03
#define ISCSI_TASK_ATTR_ACA 0x04
#define SCSI_RSP_PDU_FLAG_BI_READ_OV
#define SCSI_RSP_PDU_FLAG_BI_READ_UN
#define SCSI_RSP_PDU_FLAG_OVERFLOW
#define SCSI_RSP_PDU_FLAG_UNDERFLOW
#define ISCSI_SERVICE_RSP_COMMAND_C
#define ISCSI_SERVICE_RSP_TARGET_FAIL
#define ISCSI_TMF_RSP_PDU_RSP_FUNCTI
#define ISCSI_TMF_RSP_PDU_RSP_TASK_N
#define ISCSI_TMF_RSP_PDU_RSP_LUN_NO
#define ISCSI_TMF_RSP_PDU_RSP_TASK_S
#define ISCSI_TMF_RSP_PDU_RSP_TASK_R
#define ISCSI_TMF_RSP_PDU_RSP_NOT_SU
#define ISCSI_TMF_RSP_PDU_RSP_FUNCTI
#define ISCSI_TMF_RSP_PDU_RSP_FUNCTI
#define SCSI_DATA_IN_PDU_FLAG_ACKKN
#define SCSI_DATA_IN_PDU_FLAG_OVERF
#define SCSI_DATA_IN_PDU_FLAG_UNDER
#define SCSI_DATA_IN_PDU_FLAG_STATUS
#define ISCSI_LOGIN_REQ_PDU_FLAG_TRA
#define ISCSI_LOGIN_REQ_PDU_FLAG_CON
```

#define ISCSI_LOGIN_RSP_PDU_FLAG_TRANSIT IS
#define ISCSI_LOGIN_RSP_PDU_FLAG_CONTINUE
#define ISCSI_LOGIN_STATUS_SUCCESS 0
#define ISCSI_LOGIN_STATUS_REDIRECTION 1
#define ISCSI_LOGIN_STATUS_INITIATOR_ERROR
#define ISCSI_LOGIN_STATUS_TARGET_ERROR 3
#define ISCSI_LOGOUT_REASON_CLOSE_SESSION
#define ISCSI_LOGOUT_REASON_CLOSE_CONNECT
#define ISCSI_LOGOUT_REASON_REMOVE_CONNE
#define ISCSI_LOGOUT_RESPONSE_SESSION_CLOS
#define ISCSI_LOGOUT_RESPONSE_CID_NOT_FOU
#define ISCSI_LOGOUT_RESPONSE_RECOVERY_N
#define ISCSI_LOGOUT_RESPONSE_CLEANUP_FAI
#define ISCSI_SNACK_REQUEST_TYPE_DATA_OR_
#define ISCSI_SNACK_REQUEST_TYPE_STATUS 1
#define ISCSI_SNACK_REQUEST_TYPE_DATA_ACK
#define ISCSI_SNACK_REQUEST_TYPE_RDATA 3
#define ISCSI_SECURITY_NEGOTIATION 0
#define ISCSI_LOGIN_OPERATIONAL_NEGOTIATIO
#define ISCSI_FULL_FEATURE_PHASE 3

Typedefs

typedef struct _ISCSI_SESSION ISCSI_SESSION
typedef struct _ISCSI_CONNECTION ISCSI_CONNECTION
typedef struct _ISCSI_BASIC_HEADER ISCSI_BASIC_HEADER
typedef struct _ISCSI_ADDTIONAL_HEADER ISCSI_ADDITIONAL_HEADER
typedef struct _ISCSI_BI_EXP_READ_DATA_LEN_AHS ISCSI_BI_EXP_READ_DATA_LEN_AHS
typedef struct _SCSI_COMMAND SCSI_COMMAND
typedef struct _SCSI_RESPONSE SCSI_RESPONSE
typedef struct _ISCSI_SENSE_DATA ISCSI_SENSE_DATA
typedef struct _ISCSI_TMF_REQUEST ISCSI_TMF_REQUEST
typedef struct _ISCSI_TMF_RESPONSE ISCSI_TMF_RESPONSE
typedef struct _ISCSI_SCSI_DATA_OUT ISCSI_SCSI_DATA_OUT
typedef struct _ISCSI_SCSI_DATA_IN ISCSI_SCSI_DATA_IN
typedef struct _ISCSI_READY_TO_TRANSFER ISCSI_READY_TO_TRANSFER
typedef struct _ISCSI_ASYNC_MESSAGE ISCSI_ASYNC_MESSAGE
typedef struct _ISCSI_LOGIN_REQUEST ISCSI_LOGIN_REQUEST
typedef struct _ISCSI_LOGIN_RESPONSE ISCSI_LOGIN_RESPONSE
typedef struct _ISCSI_LOGOUT_REQUEST ISCSI_LOGOUT_REQUEST
typedef struct _ISCSI_LOGOUT_RESPONSE ISCSI_LOGOUT_RESPONSE
typedef struct _ISCSI_SNACK_REQUEST ISCSI_SNACK_REQUEST
typedef struct _ISCSI_REJECT ISCSI_REJECT
typedef struct _ISCSI_NOP_OUT ISCSI_NOP_OUT
typedef struct _ISCSI_NOP_IN ISCSI_NOP_IN
typedef struct _ISCSI_XFER_CONTEXT ISCSI_XFER_CONTEXT
typedef struct _ISCSI_IN_BUFFER_CONTEXT ISCSI_IN_BUFFER_CONTEXT

typedef struct _ISCSI_TCB ISCSI_TCB
typedef struct _ISCSI_KEY_VALUE_PAIR ISCSI_KEY_VALUE_PAIR

Enumerations

enum DATA_DIRECTION { DataIn = 0, DataC
enum ISCSI_DIGEST_TYPE { IScsiDigestNon

Functions

VOID IScsiAttatchConnection (IN OUT ISCSI_ *Conn)
VOID IScsiDetatchConnection (IN OUT ISCSI_
EFI_STATUS IScsiConnLogin (IN OUT ISCSI_CONNI
ISCSI_CONNECTION * IScsiCreateConnection (IN ISCSI_SESSI
VOID IScsiDestroyConnection (IN ISCSI_CON
EFI_STATUS IScsiSessionLogin (IN ISCSI_SESSION *
EFI_STATUS IScsiSessionReLogin (IN ISCSI_SESSIO
EFI_STATUS IScsiSendLoginReq (IN ISCSI_CONNEC
EFI_STATUS IScsiReceiveLoginRsp (IN ISCSI_CONN
EFI_STATUS IScsiAddKeyValuePair (IN OUT NET_B
NET_BUF * IScsiPrepareLoginReq (IN OUT ISCSI_C
EFI_STATUS IScsiProcessLoginRsp (IN OUT ISCSI_C
EFI_STATUS IScsiUpdateTargetAddress (IN OUT ISCS Len)
VOID EFIAPI IScsiFreeNbufList (VOID *Arg)
EFI_STATUS IScsiReceivePdu (IN ISCSI_CONNECTI ISCSI_IN_BUFFER_CONTEXT *Contex DataDigest, IN EFI_EVENT TimeoutEve
EFI_STATUS IScsiCheckOpParams (IN OUT ISCSI_C
VOID IScsiFillOpParams (IN ISCSI_CONNECT
EFI_STATUS IScsiPadSegment (IN OUT NET_BUF *P
LIST_ENTRY * IScsiBuildKeyValueList (IN CHAR8 *Da
CHAR8 * IScsiGetValueByKeyFromList (IN OUT I
VOID IScsiFreeKeyValueList (IN LIST_ENTRY
EFI_STATUS IScsiNormalizeName (IN OUT CHAR8 *
EFI_STATUS IScsiExecuteScsiCommand (IN EFI_EXT UINT8 *Target, IN UINT64 Lun, IN OUT EFI_EXT_SCSI_PASS_THRU_SCSI_RE
EFI_STATUS IScsiSessionReinstatement (IN ISCSI_SE
VOID IScsiSessionInit (IN OUT ISCSI_SESSIO
VOID IScsiSessionAbort (IN OUT ISCSI_SESS

Detailed Description

The header file of iSCSI Protocol that defines many specific data structures.

Copyright (c) 2004 - 2014, Intel Corporation. All rights reserved.
This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

#define CONN_STATE_FREE 0

connection state for initiator

Referenced by IScsiCreateConnection().

#define CONN_STATE_IN_CLEANUP 7

#define CONN_STATE_IN_LOGIN 2

Referenced by IScsiConnLogin().

#define CONN_STATE_IN_LOGOUT 4

#define CONN_STATE_LOGGED_IN 3

#define CONN_STATE_LOGOUT_REQUESTED 5

#define CONN_STATE_XPT_WAIT 1

#define DEFAULT_MAX_OUTSTANDING_R2T 1

Referenced by IScsiSessionInit().

#define DEFAULT_MAX_RECV_DATA_SEG_LEN 8192

Referenced by IScsiCreateConnection(), and IScsiPrepareLoginReq().

#define HTON24 (Dst,
Src)

Value:

```
do { \
    (Dst)[0]  = (UINT8) ((UINT8) ((Src) >> 16) & 0xFF); \
    (Dst)[1]  = (UINT8) ((UINT8) ((Src) >> 8) & 0xFF); \
    (Dst)[2]  = (UINT8) ((UINT8) (Src) & 0xFF); \
  } while (0);
```

#define ISCSI_AHS_TYPE_BI_EXP_READ_DATA_LEN 0x2

Referenced by IScsiNewScsiCmdPdu().

#define ISCSI_AHS_TYPE_EXT_CDB 0x1

Referenced by IScsiNewScsiCmdPdu().

#define ISCSI_BHS_FLAG_FINAL 0x80

Referenced by IScsiGenerateDataOutPduSequence(), IScsiNewScsiCmdPdu(), and IScsiOnDataInRcvd().

#define ISCSI_CHECK_OPCODE (PduHdr, Op) ((((PduHdr)->OpCode) & ISCSI_OPCODE_MASK) == (Op))

Referenced by IScsiProcessLoginRsp().

#define ISCSI_CLEAR_FLAG (PduHdr, Flag) (((ISCSI_BASIC_HEADER *) (PduHdr))->Flags &= ~(Flag))

#define ISCSI_CMD_PDU_TASK_ATTR_MASK 0x07

#define ISCSI_FLAG_ON (PduHdr, Flag) ((BOOLEAN) ((((ISCSI_BASIC_HEADER *) (PduHdr))->Flags & (Flag)) == (Flag)))

Referenced by IScsiOnDataInRcvd(), IScsiOnScsiRspRcvd(), IScsiPrepareLoginReq(), and IScsiProcessLoginRsp().

#define ISCSI_FULL_FEATURE_PHASE 3

Referenced by IScsiConnLogin(), and IScsiProcessLoginRsp().

#define ISCSI_GET_BUFFER_OFFSET (PduHdr) NTOHL (((ISCSI_SCSI_DATA_IN *) (PduHdr))->BufferOffset)

Referenced by IScsiReceivePdu().

```
#define ISCSI_GET_CURRENT_STAGE ( PduHdr ) ((UINT8) (((PduHdr)->Flags >> 2) & 0x3))
```

Referenced by IScsiProcessLoginRsp().

```
#define ISCSI_GET_DATASEG_LEN ( PduHdr ) NTOH24 (((ISCSI_BASIC_HEADER *) (PduHdr))->DataSegmentLength)
```

Referenced by IScsiExecuteScsiCommand(), IScsiOnScsiRspRcvd(), IScsiPrepareLoginReq(), IScsiProcessLoginRsp(), and IScsiReceivePdu().

```
#define ISCSI_GET_NEXT_STAGE ( PduHdr ) ((UINT8) (((PduHdr)->Flags) & 0x3))
```

Referenced by IScsiProcessLoginRsp().

```
#define ISCSI_GET_OPCODE ( PduHdr ) ((((ISCSI_BASIC_HEADER *) (PduHdr))->OpCode) & ISCSI_OPCODE_MASK)
```

Referenced by IScsiExecuteScsiCommand(), and IScsiReceivePdu().

```
#define ISCSI_GET_PAD_LEN ( DataLen ) ((~(DataLen) + 1) & 0x3)
```

Referenced by IScsiNewDataSegment(), IScsiPadSegment(), and IScsiReceivePdu().

```
#define ISCSI_IMMEDIATE_ON ( PduHdr ) ((PduHdr)->OpCode & ISCSI_REQ_IMMEDIATE)
```

```
#define ISCSI_KEY_AUTH_METHOD "AuthMethod"
```

Referenced by IScsiCHAPOnRspReceived(), and IScsiCHAPToSendReq().

```
#define ISCSI_KEY_DATA_DIGEST "DataDigest"
```

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

```
#define ISCSI_KEY_DATA_PDU_IN_ORDER "DataPDUInOrder"
```

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

```
#define ISCSI_KEY_DATA_SEQUENCE_IN_ORDER "DataSequenceInOrder"
```

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

#define ISCSI_KEY_DEFAULT_TIME2RETAIN "DefaultTime2Retain"

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

#define ISCSI_KEY_DEFAULT_TIME2WAIT "DefaultTime2Wait"

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

#define ISCSI_KEY_ERROR_RECOVERY_LEVEL "ErrorRecoveryLevel"

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

#define ISCSI_KEY_FIRST_BURST_LENGTH "FirstBurstLength"

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

#define ISCSI_KEY_HEADER_DIGEST "HeaderDigest"

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

#define ISCSI_KEY_IMMEDIATE_DATA "ImmediateData"

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

#define ISCSI_KEY_INITIAL_R2T "InitialR2T"

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

#define ISCSI_KEY_INITIATOR_ALIAS "InitiatorAlias"

#define ISCSI_KEY_INITIATOR_NAME "InitiatorName"

Referenced by IScsiCHAPToSendReq().

#define ISCSI_KEY_MAX_BURST_LENGTH "MaxBurstLength"

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

#define ISCSI_KEY_MAX_CONNECTIONS "MaxConnections"

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

#define ISCSI_KEY_MAX_OUTSTANDING_R2T "MaxOutstandingR2T"

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

#define ISCSI_KEY_MAX_RECV_DATA_SEGMENT_LENGTH "MaxRecvDataSegmentLength"

Referenced by IScsiCheckOpParams(), and IScsiFillOpParams().

#define ISCSI_KEY_SESSION_TYPE "SessionType"

Referenced by IScsiCHAPToSendReq(), and IScsiCheckOpParams().

#define ISCSI_KEY_TARGET_ADDRESS "TargetAddress"

Referenced by IScsiUpdateTargetAddress().

#define ISCSI_KEY_TARGET_ALIAS "TargetAlias"

Referenced by IScsiCheckOpParams().

#define ISCSI_KEY_TARGET_NAME "TargetName"

Referenced by IScsiCHAPToSendReq().

#define ISCSI_KEY_TARGET_PORTAL_GROUP_TAG "TargetPortalGroupTag"

Referenced by IScsiCHAPOnRspReceived(), and IScsiCheckOpParams().

#define ISCSI_KEY_VALUE_NONE "None"

Referenced by IScsiCHAPOnRspReceived(), IScsiCHAPToSendReq(), and IScsiCheckOpParams().

#define ISCSI_LOGIN_OPERATIONAL_NEGOTIATION 1

Referenced by IScsiCreateConnection(), IScsiPrepareLoginReq(), and IScsiProcessLoginRsp().

#define ISCSI_LOGIN_REQ_PDU_FLAG_CONTINUE 0x40

#define ISCSI_LOGIN_REQ_PDU_FLAG_TRANSIT 0x80

Referenced by IScsiCHAPToSendReq(), and IScsiPrepareLoginReq().

#define ISCSI_LOGIN_RSP_PDU_FLAG_CONTINUE ISCSI_LOGIN_REQ_PDU_FLAG_CONTINUE

Referenced by IScsiProcessLoginRsp().

#define ISCSI_LOGIN_RSP_PDU_FLAG_TRANSIT ISCSI_LOGIN_REQ_PDU_FLAG_TRANSIT

Referenced by IScsiProcessLoginRsp().

#define ISCSI_LOGIN_STATUS_INITIATOR_ERROR 2

#define ISCSI_LOGIN_STATUS_REDIRECTION 1

Referenced by IScsiProcessLoginRsp().

#define ISCSI_LOGIN_STATUS_SUCCESS 0

Referenced by IScsiProcessLoginRsp().

#define ISCSI_LOGIN_STATUS_TARGET_ERROR 3

#define ISCSI_LOGOUT_REASON_CLOSE_CONNECTION 1

#define ISCSI_LOGOUT_REASON_CLOSE_SESSION 0

#define ISCSI_LOGOUT_REASON_REMOVE_CONNECTION_FOR_RECOVERY 2

#define ISCSI_LOGOUT_RESPONSE_CID_NOT_FOUND 1

#define ISCSI_LOGOUT_RESPONSE_CLEANUP_FAILED 3

#define ISCSI_LOGOUT_RESPONSE_RECOVERY_NOT_SUPPORTED 2

#define ISCSI_LOGOUT_RESPONSE_SESSION_CLOSED_SUCCESS 0

#define ISCSI_MAX_CONNS_PER_SESSION 1

Referenced by IScsiSessionInit().

#define ISCSI_OPCODE_ASYNC_MSG 0x32

Referenced by IScsiReceivePdu().

#define ISCSI_OPCODE_LOGIN_REQ 0x03

Referenced by IScsiPrepareLoginReq().

#define ISCSI_OPCODE_LOGIN_RSP 0x23

Referenced by IScsiProcessLoginRsp(), and IScsiReceivePdu().

#define ISCSI_OPCODE_LOGOUT_REQ 0x06

#define ISCSI_OPCODE_LOGOUT_RSP 0x26

#define ISCSI_OPCODE_MASK 0x3F

#define ISCSI_OPCODE_NOP_IN 0x20

Referenced by IScsiExecuteScsiCommand(), and IScsiReceivePdu().

#define ISCSI_OPCODE_NOP_OUT 0x00

#define ISCSI_OPCODE_R2T 0x31

Referenced by IScsiExecuteScsiCommand().

#define ISCSI_OPCODE_REJECT 0x3f

Referenced by IScsiReceivePdu().

#define ISCSI_OPCODE_SCSI_CMD 0x01

Referenced by IScsiNewScsiCmdPdu().

#define ISCSI_OPCODE_SCSI_DATA_IN 0x25

Referenced by IScsiExecuteScsiCommand(), and IScsiReceivePdu().

#define ISCSI_OPCODE_SCSI_DATA_OUT 0x05

Referenced by IScsiNewDataOutPdu().

#define ISCSI_OPCODE_SCSI_RSP 0x21

Referenced by IScsiExecuteScsiCommand(), and IScsiReceivePdu().

#define ISCSI_OPCODE_SCSI_TMF_REQ 0x02

#define ISCSI_OPCODE_SCSI_TMF_RSP 0x22

#define ISCSI_OPCODE_SNACK_REQ 0x10

#define ISCSI_OPCODE_TEXT_REQ 0x04

#define ISCSI_OPCODE_TEXT_RSP 0x24

Referenced by IScsiReceivePdu().

#define ISCSI_OPCODE_VENDOR_I0 0x1c

#define ISCSI_OPCODE_VENDOR_I1 0x1d

#define ISCSI_OPCODE_VENDOR_I2 0x1e

#define ISCSI_OPCODE_VENDOR_T0 0x3c

Referenced by IScsiExecuteScsiCommand(), and IScsiReceivePdu().

#define ISCSI_OPCODE_VENDOR_T1 0x3d

Referenced by IScsiExecuteScsiCommand(), and IScsiReceivePdu().

#define ISCSI_OPCODE_VENDOR_T2 0x3e

Referenced by IScsiExecuteScsiCommand(), and IScsiReceivePdu().

#define ISCSI_REQ_IMMEDIATE 0x40

Referenced by IScsiPrepareLoginReq().

#define ISCSI_RESERVED_TAG 0xffffffff

Referenced by IScsiExecuteScsiCommand(), IScsiNewDataOutPdu(), and IScsiOnNopInRcvd().

#define ISCSI_ROUNDUP (DataLen) (((DataLen) + 3) &~(0x3))

Referenced by IScsiNewScsiCmdPdu().

#define ISCSI_SECURITY_NEGOTIATION 0

Referenced by IScsiCHAPOnRspReceived(), IScsiCHAPToSendReq(), IScsiCreateConnection(), IScsiPrepareLoginReq(), and IScsiProcessLoginRsp().

#define ISCSI_SEQ_EQ (s1,
s2) ((s1) == (s2))

Referenced by IScsiCheckSN(), and IScsiOnR2TRcvd().

#define ISCSI_SEQ_GT (s1,
s2)

Value:

```
( \
      (((INT32) (s1) < (INT32) (s2)) && (s2 - s1) > ((UINT32) 1 << 31)) || \
      (((INT32) (s1) > (INT32) (s2)) && (s1 - s2) < ((UINT32) 1 << 31)) \
    )
```

Referenced by IScsiNewTcb(), and IScsiUpdateCmdSN().

#define ISCSI_SEQ_LT (s1,
s2)

Value:

```
( \
      (((INT32) (s1) < (INT32) (s2)) && (s2 - s1) < ((UINT32) 1 << 31)) || \
      (((INT32) (s1) > (INT32) (s2)) && (s1 - s2) > ((UINT32) 1 << 31)) \
    )
```

Referenced by IScsiCheckSN(), and IScsiUpdateCmdSN().

#define ISCSI_SERVICE_RSP_COMMAND_COMPLETE_AT_TARGET 0x00

Referenced by IScsiOnScsiRspRcvd().

```
#define ISCSI_SERVICE_RSP_TARGET_FAILURE   0x01
```

```
#define
ISCSI_SET_DATASEG_LEN        ( PduHdr,
                               Len        )  HTON24 (((ISCSI_BASIC_HEADER *)
                                             (PduHdr))->DataSegmentLength, (Len))
```

Referenced by IScsiAddKeyValuePair(), IScsiNewDataOutPdu(), and IScsiNewScsiCmdPdu().

```
#define
ISCSI_SET_FLAG        ( PduHdr,
                        Flag       )  (((ISCSI_BASIC_HEADER *) (PduHdr))->Flags |=
                                      (BOOLEAN)(Flag))
```

Referenced by IScsiCHAPToSendReq(), IScsiGenerateDataOutPduSequence(), IScsiNewScsiCmdPdu(), and IScsiPrepareLoginReq().

```
#define
ISCSI_SET_OPCODE        ( PduHdr,
                          Op,
                          Flgs       )  ((((ISCSI_BASIC_HEADER *) (PduHdr))->OpCode) = ((Op)
                                        | (Flgs)))
```

Referenced by IScsiNewDataOutPdu(), IScsiNewScsiCmdPdu(), and IScsiPrepareLoginReq().

```
#define
ISCSI_SET_STAGES        ( PduHdr,
                          Cur,
                          Nxt        )  ((PduHdr)->Flags = (UINT8) ((PduHdr)->Flags | ((Cur) << 2 |
                                        (Nxt))))
```

Referenced by IScsiPrepareLoginReq().

```
#define ISCSI_SNACK_REQUEST_TYPE_DATA_ACK   2
```

```
#define ISCSI_SNACK_REQUEST_TYPE_DATA_OR_R2T   0
```

```
#define ISCSI_SNACK_REQUEST_TYPE_RDATA   3
```

```
#define ISCSI_SNACK_REQUEST_TYPE_STATUS   1
```

```
#define ISCSI_TASK_ATTR_ACA   0x04
```

#define ISCSI_TASK_ATTR_HOQ 0x03

#define ISCSI_TASK_ATTR_ORDERD 0x02

#define ISCSI_TASK_ATTR_SIMPLE 0x01

Referenced by IScsiNewScsiCmdPdu().

#define ISCSI_TASK_ATTR_UNTAGGED 0x00

#define ISCSI_TMF_RSP_PDU_RSP_FUNCTION_AHTH_FAILED 6

#define ISCSI_TMF_RSP_PDU_RSP_FUNCTION_COMPLETE 0

#define ISCSI_TMF_RSP_PDU_RSP_FUNCTION_REJECTED 255

#define ISCSI_TMF_RSP_PDU_RSP_LUN_NOT_EXIST 2

#define ISCSI_TMF_RSP_PDU_RSP_NOT_SUPPORTED 5

#define ISCSI_TMF_RSP_PDU_RSP_TASK_NOT_EXIST 1

#define ISCSI_TMF_RSP_PDU_RSP_TASK_REASSGIN_NOT_SUPPORTED 4

#define ISCSI_TMF_RSP_PDU_RSP_TASK_STILL_ALLEGIANT 3

#define ISCSI_VERSION_MAX 0x00

Referenced by IScsiPrepareLoginReq().

#define ISCSI_VERSION_MIN 0x00

Referenced by IScsiPrepareLoginReq().

#define ISCSI_WELL_KNOWN_PORT 3260

Referenced by IScsiConfigProcessDefault(), IScsiConvertIfrNvDataToAttemptConfigData(), IScsiDhcp6ExtractRootPath(), IScsiDhcpExtractRootPath(), and IScsiUpdateTargetAddress().

#define MAX_RECV_DATA_SEG_LEN_IN_FFP 65536

Referenced by IScsiFillOpParams(), and IScsiSessionInit().

#define NTOH24 (src) (((src)[0] << 16) | ((src)[1] << 8) | ((src)[2]))

Referenced by IScsiAddKeyValuePair().

#define SCSI_CMD_PDU_FLAG_READ 0x40

Referenced by IScsiNewScsiCmdPdu().

#define SCSI_CMD_PDU_FLAG_WRITE 0x20

Referenced by IScsiNewScsiCmdPdu().

#define SCSI_DATA_IN_PDU_FLAG_ACKKNOWLEDGE 0x40

#define SCSI_DATA_IN_PDU_FLAG_OVERFLOW SCSI_RSP_PDU_FLAG_OVERFLOW

Referenced by IScsiOnDataInRcvd().

#define SCSI_DATA_IN_PDU_FLAG_STATUS_VALID 0x01

Referenced by IScsiOnDataInRcvd().

#define SCSI_DATA_IN_PDU_FLAG_UNDERFLOW SCSI_RSP_PDU_FLAG_UNDERFLOW

Referenced by IScsiOnDataInRcvd().

#define SCSI_RSP_PDU_FLAG_BI_READ_OVERFLOW 0x10

Referenced by IScsiOnScsiRspRcvd().

#define SCSI_RSP_PDU_FLAG_BI_READ_UNDERFLOW 0x08

Referenced by IScsiOnScsiRspRcvd().

#define SCSI_RSP_PDU_FLAG_OVERFLOW 0x04

Referenced by IScsiOnDataInRcvd(), and IScsiOnScsiRspRcvd().

#define SCSI_RSP_PDU_FLAG_UNDERFLOW 0x02

Referenced by IScsiOnDataInRcvd(), and IScsiOnScsiRspRcvd().

#define SESSION_STATE_FAILED 2

Referenced by IScsiSessionAbort().

#define SESSION_STATE_FREE 0
session state for initiator

Referenced by IScsiProcessLoginRsp(), IScsiSessionInit(), and IScsiSessionReinstatement().

#define SESSION_STATE_LOGGED_IN 1

Referenced by IScsiExecuteScsiCommand(), IScsiGetTcpConnDevicePath(), IScsiSessionAbort(), and IScsiSessionLogin().

Typedef Documentation

typedef struct _ISCSI_ADDTIONAL_HEADER ISCSI_ADDITIONAL_HEADER

typedef struct _ISCSI_ASYNC_MESSAGE ISCSI_ASYNC_MESSAGE

typedef struct _ISCSI_BASIC_HEADER ISCSI_BASIC_HEADER
iSCSI Basic Header Segment

typedef struct _ISCSI_BI_EXP_READ_DATA_LEN_AHS ISCSI_BI_EXP_READ_DATA_LEN_AHS

typedef struct _ISCSI_CONNECTION ISCSI_CONNECTION

typedef struct _ISCSI_IN_BUFFER_CONTEXT ISCSI_IN_BUFFER_CONTEXT

typedef struct _ISCSI_KEY_VALUE_PAIR ISCSI_KEY_VALUE_PAIR

typedef struct _ISCSI_LOGIN_REQUEST ISCSI_LOGIN_REQUEST
Login Request.

typedef struct _ISCSI_LOGIN_RESPONSE ISCSI_LOGIN_RESPONSE
Login Response.

typedef struct _ISCSI_LOGOUT_REQUEST ISCSI_LOGOUT_REQUEST
Logout Request.

typedef struct _ISCSI_LOGOUT_RESPONSE ISCSI_LOGOUT_RESPONSE
Logout Response.

typedef struct _ISCSI_NOP_IN ISCSI_NOP_IN
NOP-In.

typedef struct _ISCSI_NOP_OUT ISCSI_NOP_OUT
NOP-Out.

typedef struct _ISCSI_READY_TO_TRANSFER ISCSI_READY_TO_TRANSFER
Ready To Transfer.

typedef struct _ISCSI_REJECT ISCSI_REJECT
Reject.

typedef struct _ISCSI_SCSI_DATA_IN ISCSI_SCSI_DATA_IN
SCSI Data-In

typedef struct _ISCSI_SCSI_DATA_OUT ISCSI_SCSI_DATA_OUT
SCSI Data-Out

typedef struct _ISCSI_SENSE_DATA ISCSI_SENSE_DATA

typedef struct _ISCSI_SESSION ISCSI_SESSION

typedef struct _ISCSI_SNACK_REQUEST ISCSI_SNACK_REQUEST
SNACK Request.

typedef struct _ISCSI_TCB ISCSI_TCB

typedef struct _ISCSI_TMF_REQUEST ISCSI_TMF_REQUEST
iSCSI Task Managment Function Request.

typedef struct _ISCSI_TMF_RESPONSE ISCSI_TMF_RESPONSE
iSCSI Task Management Function Response.

typedef struct _ISCSI_XFER_CONTEXT ISCSI_XFER_CONTEXT

typedef struct _SCSI_COMMAND SCSI_COMMAND
SCSI Command

typedef struct _SCSI_RESPONSE SCSI_RESPONSE
SCSI Response

Enumeration Type Documentation

enum DATA_DIRECTION

Enumerator:

DataIn
DataOut
DataBi

enum ISCSI_DIGEST_TYPE

Enumerator:

IScsiDigestNone
IScsiDigestCRC32

Function Documentation

```
EFI_STATUS IScsiAddKeyValuePair ( IN OUT NET_BUF *   Pdu,
                                  IN CHAR8 *         Key,
                                  IN CHAR8 *         Value
                                )
```

Add an iSCSI key-value pair as a string into the data segment of the Login Request PDU. The DataSegmentLength and the actual size of the net buffer containing this PDU will be updated.

Parameters:

[in,out]	*Pdu*	The iSCSI PDU whose data segment the key-value pair will be added to.
[in]	*Key*	The key name string.
[in]	*Value*	The value string.

Return values:

EFI_SUCCESS	The key-valu pair is added to the PDU's datasegment and the correspondence length fields are updated.
EFI_OUT_OF_RESOURCES	There is not enough space in the PDU to add the key-value pair.

Add an iSCSI key-value pair as a string into the data segment of the Login Request PDU. The DataSegmentLength and the actual size of the net buffer containing this PDU will be updated.

Parameters:

`[in,out]`	*Pdu*	The iSCSI PDU whose data segment the key-value pair will be added to.
`[in]`	*Key*	The key name string.
`[in]`	*Value*	The value string.

Return values:

EFI_SUCCESS	The key-value pair is added to the PDU's data segment and the correspondence length fields are updated.
EFI_OUT_OF_RESOURCES	There is not enough space in the PDU to add the key-value pair.
EFI_PROTOCOL_ERROR	There is no such data in the net buffer.

References _ISCSI_LOGIN_REQUEST::DataSegmentLength, ISCSI_SET_DATASEG_LEN, and NTOH24.

Referenced by IScsiCHAPToSendReq(), and IScsiFillOpParams().

```
VOID IScsiAttatchConnection ( IN OUT ISCSI_SESSION *      Session,
                              IN OUT ISCSI_CONNECTION *   Conn
                            )
```

Attach the iSCSI connection to the iSCSI session.

Parameters:

`[in,out]`	*Session*	The iSCSI session.
`[in,out]`	*Conn*	The iSCSI connection.

Referenced by IScsiSessionLogin().

```
LIST_ENTRY* IScsiBuildKeyValueList ( IN CHAR8 *   Data,
                                     IN UINT32    Len
                                   )
```

Build a key-value list from the data segment.

Parameters:

`[in]`	*Data*	The data segment containing the key-value pairs.
`[in]`	*Len*	Length of the data segment.

Returns:

The key-value list.

Return values:

NULL	Other errors as indicated.

References IScsiFreeKeyValueList(), _ISCSI_KEY_VALUE_PAIR::Key,

_ISCSI_KEY_VALUE_PAIR::List, and _ISCSI_KEY_VALUE_PAIR::Value.

Referenced by IScsiCHAPOnRspReceived(), IScsiCheckOpParams(), and IScsiUpdateTargetAddress().

EFI_STATUS IScsiCheckOpParams (IN OUT ISCSI_CONNECTION * *Conn*)

Check and get the result of the parameter negotiation.

Parameters:

`[in,out]` *Conn* The connection in iSCSI login.

Return values:

EFI_SUCCESS	The parmeter check is passed and negotiation is finished.
EFI_PROTOCOL_ERROR	Some kind of iSCSI protocol error occurred.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.

References _ISCSI_SESSION::DataPDUInOrder, _ISCSI_SESSION::DataSequenceInOrder, _ISCSI_SESSION::DefaultTime2Retain, _ISCSI_SESSION::DefaultTime2Wait, _ISCSI_SESSION::ErrorRecoveryLevel, _ISCSI_SESSION::FirstBurstLength, _ISCSI_SESSION::ImmediateData, _ISCSI_SESSION::InitialR2T, ISCSI_KEY_DATA_DIGEST, ISCSI_KEY_DATA_PDU_IN_ORDER, ISCSI_KEY_DATA_SEQUENCE_IN_ORDER, ISCSI_KEY_DEFAULT_TIME2RETAIN, ISCSI_KEY_DEFAULT_TIME2WAIT, ISCSI_KEY_ERROR_RECOVERY_LEVEL, ISCSI_KEY_FIRST_BURST_LENGTH, ISCSI_KEY_HEADER_DIGEST, ISCSI_KEY_IMMEDIATE_DATA, ISCSI_KEY_INITIAL_R2T, ISCSI_KEY_MAX_BURST_LENGTH, ISCSI_KEY_MAX_CONNECTIONS, ISCSI_KEY_MAX_OUTSTANDING_R2T, ISCSI_KEY_MAX_RECV_DATA_SEGMENT_LENGTH, ISCSI_KEY_SESSION_TYPE, ISCSI_KEY_TARGET_ALIAS, ISCSI_KEY_TARGET_PORTAL_GROUP_TAG, ISCSI_KEY_VALUE_NONE, IScsiBuildKeyValueList(), IScsiDigestCRC32, IScsiDigestNone, IScsiFreeKeyValueList(), IScsiGetValueByKeyFromList(), IScsiNetNtoi(), _ISCSI_SESSION::MaxBurstLength, _ISCSI_SESSION::MaxConnections, and _ISCSI_SESSION::MaxOutstandingR2T.

Referenced by IScsiProcessLoginRsp().

EFI_STATUS IScsiConnLogin (IN OUT ISCSI_CONNECTION * *Conn*,
IN UINT16 *Timeout*
)

This function performs the iSCSI connection login.

Parameters:

`[in,out]` *Conn* The iSCSI connection to login.
Timeout The timeout value in milliseconds.

Return values:

EFI_SUCCESS	The iSCSI connection is logged into the iSCSI target.
EFI_TIMEOUT	Timeout occurred during the login procedure.
Others	Other errors as indicated.

This function does the iSCSI connection login.

Parameters:

`[in,out]` *Conn* The iSCSI connection to login.

Timeout The timeout value in millisecond.

Return values:

EFI_SUCCESS The iSCSI connection is logged into the iSCSI target.
EFI_TIMEOUT Timeout occurred during the login procedure.
Others Other errors as indicated.

References CONN_STATE_IN_LOGIN, ISCSI_FULL_FEATURE_PHASE, IScsiReceiveLoginRsp(), and IScsiSendLoginReq().

Referenced by IScsiSessionLogin().

ISCSI_CONNECTION* IScsiCreateConnection (IN ISCSI_SESSION * *Session*)

Create a TCP connection for the iSCSI session.

Parameters:

[in] *Session* Points to the iSCSI session.

Returns:

The newly created iSCSI connection.

References _ISCSI_CONNECTION::AuthStep, _ISCSI_CONNECTION::Cid, CONN_STATE_FREE, _ISCSI_DRIVER_DATA::Controller, _ISCSI_CONNECTION::CurrentStage, _ISCSI_CONNECTION::DataDigest, DEFAULT_MAX_RECV_DATA_SEG_LEN, _ISCSI_CONNECTION::ExpStatSN, _ISCSI_SESSION_CONFIG_NVDATA::Gateway, _ISCSI_CONNECTION::HeaderDigest, _ISCSI_DRIVER_DATA::Image, IP_MODE_AUTOCONFIG_IP6, IP_MODE_IP6, _ISCSI_SESSION_CONFIG_NVDATA::IpMode, _ISCSI_CONNECTION::Ipv6Flag, ISCSI_AUTH_INITIAL, ISCSI_CONNECTION_SIGNATURE, ISCSI_LOGIN_OPERATIONAL_NEGOTIATION, ISCSI_SECURITY_NEGOTIATION, IScsiDigestNone, _ISCSI_SESSION_CONFIG_NVDATA::LocalIp, _ISCSI_CONNECTION::MaxRecvDataSegmentLength, _ISCSI_CONNECTION::NextStage, _ISCSI_CONNECTION::ParamNegotiated, _ISCSI_CONNECTION::PartialReqSent, _ISCSI_CONNECTION::PartialRspRcvd, _ISCSI_CONNECTION::RspQue, _ISCSI_CONNECTION::Signature, _ISCSI_CONNECTION::State, _ISCSI_SESSION_CONFIG_NVDATA::SubnetMask, _ISCSI_SESSION_CONFIG_NVDATA::TargetIp, _ISCSI_SESSION_CONFIG_NVDATA::TargetPort, _ISCSI_CONNECTION::TcpIo, and _ISCSI_CONNECTION::TimeoutEvent.

Referenced by IScsiSessionLogin().

VOID IScsiDestroyConnection (IN ISCSI_CONNECTION * *Conn*)

Destroy an iSCSI connection.

Parameters:

[in] *Conn* The connection to destroy.

Referenced by IScsiSessionAbort(), and IScsiSessionLogin().

VOID IScsiDetatchConnection (IN OUT ISCSI_CONNECTION * *Conn*)

Detach the iSCSI connection from the session it belongs to.

Parameters:

[in,out]	*Conn*	The iSCSI connection.

Referenced by IScsiSessionAbort(), and IScsiSessionLogin().

```
EFI_STATUS
IScsiExecuteScsiCommand ( IN EFI_EXT_SCSI_PASS_THRU_PROTOCOL *          PassThru,
                          IN UINT8 *                                    Target,
                          IN UINT64                                     Lun,
                          IN OUT
                          EFI_EXT_SCSI_PASS_THRU_SCSI_REQUEST_PACKET    Packet
                          *
                          )
```

Execute the SCSI command issued through the EXT SCSI PASS THRU protocol.

Parameters:

[in]	*PassThru*	The EXT SCSI PASS THRU protocol.
[in]	*Target*	The target ID.
[in]	*Lun*	The LUN.
[in,out]	*Packet*	The request packet containing IO request, SCSI command buffer and buffers to read/write.

Return values:

EFI_SUCCES	The SCSI command is executed and the result is updated to the Packet.
EFI_DEVICE_ERROR	Session state was not as required.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_NOT_READY	The target can not accept new commands.
Others	Other errors as indicated.

Execute the SCSI command issued through the EXT SCSI PASS THRU protocol.

Parameters:

[in]	*PassThru*	The EXT SCSI PASS THRU protocol.
[in]	*Target*	The target ID.
[in]	*Lun*	The LUN.
[in,out]	*Packet*	The request packet containing IO request, SCSI command buffer and buffers to read/write.

Return values:

EFI_SUCCES	The SCSI command is executed and the result is updated to the Packet.
EFI_DEVICE_ERROR	Session state was not as required.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_PROTOCOL_ERROR	There is no such data in the net buffer.
EFI_NOT_READY	The target can not accept new commands.
Others	Other errors as indicated.

References _ISCSI_SESSION::Conns, _ISCSI_XFER_CONTEXT::DesiredLength, _ISCSI_SESSION::FirstBurstLength, _ISCSI_IN_BUFFER_CONTEXT::InData, _ISCSI_IN_BUFFER_CONTEXT::InDataLen, _ISCSI_SESSION::InitialR2T,

ISCSI_CONNECTION_SIGNATURE, ISCSI_DRIVER_DATA_FROM_EXT_SCSI_PASS_THRU, ISCSI_GET_DATASEG_LEN, ISCSI_GET_OPCODE, ISCSI_OPCODE_NOP_IN, ISCSI_OPCODE_R2T, ISCSI_OPCODE_SCSI_DATA_IN, ISCSI_OPCODE_SCSI_RSP, ISCSI_OPCODE_VENDOR_T0, ISCSI_OPCODE_VENDOR_T1, ISCSI_OPCODE_VENDOR_T2, ISCSI_RESERVED_TAG, IScsiDelTcb(), IScsiNewScsiCmdPdu(), IScsiNewTcb(), IScsiOnDataInRcvd(), IScsiOnNopInRcvd(), IScsiOnR2TRcvd(), IScsiOnScsiRspRcvd(), IScsiReceivePdu(), IScsiSendDataOutPduSequence(), _ISCSI_XFER_CONTEXT::Offset, _ISCSI_DRIVER_DATA::Session, SESSION_STATE_LOGGED_IN, _ISCSI_SESSION::State, _ISCSI_TCB::StatusXferd, _ISCSI_XFER_CONTEXT::TargetTransferTag, _ISCSI_CONNECTION::TcpIo, _ISCSI_CONNECTION::TimeoutEvent, and _ISCSI_TCB::XferContext.

Referenced by IScsiExtScsiPassThruFunction().

VOID IScsiFillOpParams (IN ISCSI_CONNECTION * *Conn*,
IN OUT NET_BUF * *Pdu*
)

Fill the operational parameters.

Parameters:

[in]	*Conn*	The connection in iSCSI login.
[in,out]	*Pdu*	The iSCSI login request PDU to fill the parameters.

References _ISCSI_SESSION::DataPDUInOrder, _ISCSI_SESSION::DataSequenceInOrder, _ISCSI_SESSION::DefaultTime2Retain, _ISCSI_SESSION::DefaultTime2Wait, _ISCSI_SESSION::ErrorRecoveryLevel, _ISCSI_SESSION::FirstBurstLength, _ISCSI_SESSION::ImmediateData, _ISCSI_SESSION::InitialR2T, ISCSI_KEY_DATA_DIGEST, ISCSI_KEY_DATA_PDU_IN_ORDER, ISCSI_KEY_DATA_SEQUENCE_IN_ORDER, ISCSI_KEY_DEFAULT_TIME2RETAIN, ISCSI_KEY_DEFAULT_TIME2WAIT, ISCSI_KEY_ERROR_RECOVERY_LEVEL, ISCSI_KEY_FIRST_BURST_LENGTH, ISCSI_KEY_HEADER_DIGEST, ISCSI_KEY_IMMEDIATE_DATA, ISCSI_KEY_INITIAL_R2T, ISCSI_KEY_MAX_BURST_LENGTH, ISCSI_KEY_MAX_CONNECTIONS, ISCSI_KEY_MAX_OUTSTANDING_R2T, ISCSI_KEY_MAX_RECV_DATA_SEGMENT_LENGTH, IScsiAddKeyValuePair(), IScsiDigestCRC32, MAX_RECV_DATA_SEG_LEN_IN_FFP, _ISCSI_SESSION::MaxBurstLength, _ISCSI_SESSION::MaxConnections, and _ISCSI_SESSION::MaxOutstandingR2T.

Referenced by IScsiPrepareLoginReq().

VOID IScsiFreeKeyValueList (IN LIST_ENTRY * *KeyValueList*)

Free the key-value list.

Parameters:

[in]	*KeyValueList*	The key-value list.

Referenced by IScsiBuildKeyValueList(), IScsiCHAPOnRspReceived(), IScsiCheckOpParams(), and IScsiUpdateTargetAddress().

VOID EFIAPI IScsiFreeNbufList (VOID * *Arg*)

The callback function to free the net buffer list.

Parameters:

[in] *Arg* The opaque parameter.

Referenced by IScsiGenerateDataOutPduSequence(), IScsiNewDataOutPdu(), IScsiNewScsiCmdPdu(), IScsiReceivePdu(), and IScsiSendDataOutPduSequence().

```
CHAR8* IScsiGetValueByKeyFromList ( IN OUT LIST_ENTRY *   KeyValueList,
                                    IN CHAR8 *            Key
                                  )
```

Get the value string by the key name from the key-value list. If found, the key-value entry will be removed from the list.

Parameters:

[in,out] *KeyValueList* The key-value list.
[in] *Key* The key name to find.

Returns:

The value string.

Return values:

NULL The key value pair can not be found.

Get the value string by the key name from the key-value list. If found, the key-value entry will be removed from the list.

Parameters:

[in,out] *KeyValueList* The key-value list.
[in] *Key* The key name to find.

Returns:

The value string.

Return values:

NULL The key value pair cannot be found.

References _ISCSI_KEY_VALUE_PAIR::Key, _ISCSI_KEY_VALUE_PAIR::List, and _ISCSI_KEY_VALUE_PAIR::Value.

Referenced by IScsiCHAPOnRspReceived(), IScsiCheckOpParams(), and IScsiUpdateTargetAddress().

```
EFI_STATUS IScsiNormalizeName ( IN OUT CHAR8 *   Name,
                                IN UINTN         Len
                              )
```

Normalize the iSCSI name according to RFC.

Parameters:

[in,out] *Name* The iSCSI name.
[in] *Len* length of the iSCSI name.

Return values:

EFI_SUCCESS The iSCSI name is valid and normalized.
EFI_PROTOCOL_ERROR The iSCSI name is mal-formatted or not in the IQN format.

Normalize the iSCSI name according to RFC.

Parameters:

[in,out]	*Name*	The iSCSI name.
[in]	*Len*	Length of the iSCSI name.

Return values:

EFI_SUCCESS	The iSCSI name is valid and normalized.
EFI_PROTOCOL_ERROR	The iSCSI name is malformatted or not in the IQN format.

Referenced by IScsiDhcp6ExtractRootPath(), IScsiDhcpExtractRootPath(), IScsiFormCallback(), and IScsiSetInitiatorName().

```
EFI_STATUS IScsiPadSegment ( IN OUT NET_BUF *   Pdu,
                             IN UINT32          Len
                           )
```

Pad the iSCSI AHS or data segment to an integer number of 4 byte words.

Parameters:

[in,out]	*Pdu*	The iSCSI pdu which contains segments to pad.
[in]	*Len*	The length of the last semgnet in the PDU.

Return values:

EFI_SUCCESS	The segment is padded or no need to pad it.
EFI_OUT_OF_RESOURCES	There is not enough remaining free space to add the padding bytes.

Pad the iSCSI AHS or data segment to an integer number of 4 byte words.

Parameters:

[in,out]	*Pdu*	The iSCSI pdu which contains segments to pad.
[in]	*Len*	The length of the last segment in the PDU.

Return values:

EFI_SUCCESS	The segment is padded or there is no need to pad it.
EFI_OUT_OF_RESOURCES	There is not enough remaining free space to add the padding bytes.

References ISCSI_GET_PAD_LEN.

Referenced by IScsiPrepareLoginReq().

```
NET_BUF* IScsiPrepareLoginReq ( IN OUT ISCSI_CONNECTION *   Conn   )
```

Prepare the iSCSI login request to be sent according to the current login status.

Parameters:

[in,out]	*Conn*	The connection in the iSCSI login phase.

Returns:

The pointer to the net buffer containing the iSCSI login request built.

Return values:

NULL	Other errors as indicated.

References _ISCSI_SESSION::AuthType, _ISCSI_LOGIN_REQUEST::Cid, _ISCSI_SESSION::CmdSN, _ISCSI_LOGIN_REQUEST::CmdSN, DEFAULT_MAX_RECV_DATA_SEG_LEN, _ISCSI_LOGIN_REQUEST::ExpStatSN, _ISCSI_SESSION::InitiatorTaskTag, _ISCSI_LOGIN_REQUEST::InitiatorTaskTag, ISCSI_AUTH_TYPE_KRB, ISCSI_FLAG_ON, ISCSI_GET_DATASEG_LEN, ISCSI_LOGIN_OPERATIONAL_NEGOTIATION, ISCSI_LOGIN_REQ_PDU_FLAG_TRANSIT, ISCSI_OPCODE_LOGIN_REQ, ISCSI_REQ_IMMEDIATE, ISCSI_SECURITY_NEGOTIATION, ISCSI_SET_FLAG, ISCSI_SET_OPCODE, ISCSI_SET_STAGES, ISCSI_VERSION_MAX, ISCSI_VERSION_MIN, IScsiCHAPToSendReq(), IScsiFillOpParams(), IScsiPadSegment(), _ISCSI_SESSION::Isid, _ISCSI_LOGIN_REQUEST::Isid, _ISCSI_SESSION::Tsih, _ISCSI_LOGIN_REQUEST::Tsih, _ISCSI_LOGIN_REQUEST::VersionMax, and _ISCSI_LOGIN_REQUEST::VersionMin.

Referenced by IScsiSendLoginReq().

```
EFI_STATUS IScsiProcessLoginRsp ( IN OUT ISCSI_CONNECTION *   Conn,
                                  IN OUT NET_BUF *            Pdu
                                )
```

Process the iSCSI Login Response.

Parameters:

[in,out]	*Conn*	The connection on which the iSCSI login response is received.
[in,out]	*Pdu*	The iSCSI login response PDU.

Return values:

EFI_SUCCESS	The iSCSI login response PDU is processed and all check are passed.
EFI_PROTOCOL_ERROR	Some kind of iSCSI protocol error happened.
EFI_MEDIA_CHANGED	Target is redirected.
Others	Other errors as indicated.

Process the iSCSI Login Response.

Parameters:

[in,out]	*Conn*	The connection on which the iSCSI login response is received.
[in,out]	*Pdu*	The iSCSI login response PDU.

Return values:

EFI_SUCCESS	The iSCSI login response PDU is processed, and all checks are passed.
EFI_PROTOCOL_ERROR	Some kind of iSCSI protocol error occurred.
EFI_MEDIA_CHANGED	Target is redirected.
Others	Other errors as indicated.

References _ISCSI_SESSION::AuthType, _ISCSI_SESSION::CmdSN, _ISCSI_SESSION::ExpCmdSN, _ISCSI_LOGIN_RESPONSE::ExpCmdSN, _ISCSI_SESSION::InitiatorTaskTag, _ISCSI_LOGIN_RESPONSE::InitiatorTaskTag, ISCSI_AUTH_INITIAL, ISCSI_AUTH_TYPE_KRB, ISCSI_CHECK_OPCODE, ISCSI_FLAG_ON, ISCSI_FULL_FEATURE_PHASE, ISCSI_GET_CURRENT_STAGE, ISCSI_GET_DATASEG_LEN, ISCSI_GET_NEXT_STAGE, ISCSI_LOGIN_OPERATIONAL_NEGOTIATION, ISCSI_LOGIN_RSP_PDU_FLAG_CONTINUE, ISCSI_LOGIN_RSP_PDU_FLAG_TRANSIT, ISCSI_LOGIN_STATUS_REDIRECTION, ISCSI_LOGIN_STATUS_SUCCESS, ISCSI_OPCODE_LOGIN_RSP, ISCSI_SECURITY_NEGOTIATION, IScsiCHAPOnRspReceived(), IScsiCheckOpParams(), IScsiCheckSN(), IScsiUpdateCmdSN(), IScsiUpdateTargetAddress(), _ISCSI_LOGIN_RESPONSE::Isid, _ISCSI_SESSION::Isid, _ISCSI_SESSION::MaxCmdSN, _ISCSI_LOGIN_RESPONSE::MaxCmdSN,

SESSION_STATE_FREE, _ISCSI_SESSION::State, _ISCSI_LOGIN_RESPONSE::StatSN, _ISCSI_LOGIN_RESPONSE::StatusClass, _ISCSI_LOGIN_RESPONSE::Tsih, and _ISCSI_SESSION::Tsih.

Referenced by IScsiReceiveLoginRsp().

```
EFI_STATUS IScsiReceiveLoginRsp ( IN ISCSI_CONNECTION * Conn )
```

Receive and process the iSCSI login response.

Parameters:

[in]	*Conn*	The connection in the iSCSI login phase.

Return values:

EFI_SUCCESS	The iSCSI login response PDU is received and processed.
Others	Other errors as indicated.

References IScsiProcessLoginRsp(), and IScsiReceivePdu().

Referenced by IScsiConnLogin().

```
EFI_STATUS IScsiReceivePdu ( IN ISCSI_CONNECTION *            Conn,
                             OUT NET_BUF **                   Pdu,
                             IN ISCSI_IN_BUFFER_CONTEXT *     Context,
                             IN BOOLEAN                       HeaderDigest,
                             IN BOOLEAN                       DataDigest,
                             IN EFI_EVENT                     TimeoutEvent
                           )
```

Receive an iSCSI response PDU. An iSCSI response PDU contains an iSCSI PDU header and an optional data segment. The two parts will be put into two blocks of buffers in the net buffer. The digest check will be conducted in this function if needed and the digests will be trimmed from the PDU buffer.

Parameters:

[in]	*Conn*	The iSCSI connection to receive data from.
[out]	*Pdu*	The received iSCSI pdu.
[in]	*Context*	The context used to describe information on the caller provided buffer to receive data segment of the iSCSI pdu, it's optional.
[in]	*HeaderDigest*	Whether there will be header digest received.
[in]	*DataDigest*	Whether there will be data digest.
[in]	*TimeoutEvent*	The timeout event, it's optional.

Return values:

EFI_SUCCESS	An iSCSI pdu is received.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_PROTOCOL_ERROR	Some kind of iSCSI protocol error occurred.
Others	Other errors as indicated.

Receive an iSCSI response PDU. An iSCSI response PDU contains an iSCSI PDU header and an optional data segment. The two parts will be put into two blocks of buffers in the net buffer. The digest check will be

conducted in this function if needed and the digests will be trimmed from the PDU buffer.

Parameters:

[in]	*Conn*	The iSCSI connection to receive data from.
[out]	*Pdu*	The received iSCSI pdu.
[in]	*Context*	The context used to describe information on the caller provided buffer to receive data segment of the iSCSI pdu. It is optional.
[in]	*HeaderDigest*	Whether there will be header digest received.
[in]	*DataDigest*	Whether there will be data digest.
[in]	*TimeoutEvent*	The timeout event. It is optional.

Return values:

EFI_SUCCESS	An iSCSI pdu is received.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_PROTOCOL_ERROR	Some kind of iSCSI protocol error occurred.
Others	Other errors as indicated.

References ISCSI_GET_BUFFER_OFFSET, ISCSI_GET_DATASEG_LEN, ISCSI_GET_OPCODE, ISCSI_GET_PAD_LEN, ISCSI_OPCODE_ASYNC_MSG, ISCSI_OPCODE_LOGIN_RSP, ISCSI_OPCODE_NOP_IN, ISCSI_OPCODE_REJECT, ISCSI_OPCODE_SCSI_DATA_IN, ISCSI_OPCODE_SCSI_RSP, ISCSI_OPCODE_TEXT_RSP, ISCSI_OPCODE_VENDOR_T0, ISCSI_OPCODE_VENDOR_T1, ISCSI_OPCODE_VENDOR_T2, IScsiFreeNbufList(), and IScsiNbufExtFree().

Referenced by IScsiExecuteScsiCommand(), and IScsiReceiveLoginRsp().

EFI_STATUS IScsiSendLoginReq (IN ISCSI_CONNECTION * *Conn*)

Build and send the iSCSI login request to the iSCSI target according to the current login stage.

Parameters:

[in] *Conn* The connection in the iSCSI login phase.

Return values:

EFI_SUCCESS	The iSCSI login request PDU is built and sent on this connection.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_DEVICE_ERROR	Some kind of device error happened.

Build and send the iSCSI login request to the iSCSI target according to the current login stage.

Parameters:

[in] *Conn* The connection in the iSCSI login phase.

Return values:

EFI_SUCCESS	The iSCSI login request PDU is built and sent on this connection.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_DEVICE_ERROR	Some kind of device error occurred.

References IScsiPrepareLoginReq().

Referenced by IScsiConnLogin().

VOID IScsiSessionAbort (IN OUT ISCSI_SESSION * *Session*)

Abort the iSCSI session, that is, reset all the connection and free the resources.

Parameters:

`[in,out]` *Session* The iSCSI session.

Abort the iSCSI session. That is, reset all the connection(s), and free the resources.

Parameters:

`[in,out]` *Session* The iSCSI session.

References _ISCSI_CONNECTION::Ipv6Flag, ISCSI_CONNECTION_SIGNATURE, IScsiConnReset(), IScsiDestroyConnection(), IScsiDetatchConnection(), SESSION_STATE_FAILED, SESSION_STATE_LOGGED_IN, and _ISCSI_CONNECTION::TcpIo.

Referenced by IScsiOnExitBootService(), IScsiSessionReinstatement(), IScsiStart(), and IScsiStop().

VOID IScsiSessionInit (IN OUT ISCSI_SESSION * *Session*,
IN BOOLEAN *Recovery*
)

Initialize some session parameters before login.

Parameters:

`[in,out]` *Session* The iSCSI session.
`[in]` *Recovery* Whether the request is from a fresh new start or recovery.

References DEFAULT_MAX_OUTSTANDING_R2T, ISCSI_MAX_CONNS_PER_SESSION, ISCSI_SESSION_SIGNATURE, MAX_RECV_DATA_SEG_LEN_IN_FFP, and SESSION_STATE_FREE.

Referenced by IScsiSessionReinstatement(), and IScsiStart().

EFI_STATUS IScsiSessionLogin (IN ISCSI_SESSION * *Session*)

Login the iSCSI session.

Parameters:

`[in]` *Session* The iSCSI session

Return values:

EFI_SUCCESS	The iSCSI session login procedure finished.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_NO_MEDIA	There was a media error.
Others	Other errors as indicated.

Login the iSCSI session.

Parameters:

`[in]` *Session* The iSCSI session.

Return values:

EFI_SUCCESS	The iSCSI session login procedure finished.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.

EFI_NO_MEDIA	There was a media error.
Others	Other errors as indicated.

References _ISCSI_CONNECTION::Ipv6Flag, IScsiAttatchConnection(), IScsiConnLogin(), IScsiConnReset(), IScsiCreateConnection(), IScsiDestroyConnection(), IScsiDetatchConnection(), IScsiGetIp6NicInfo(), SESSION_STATE_LOGGED_IN, and _ISCSI_CONNECTION::TcpIo.

Referenced by IScsiSessionReinstatement(), IScsiSessionReLogin(), and IScsiStart().

EFI_STATUS IScsiSessionReinstatement (IN ISCSI_SESSION * *Session*)

Reinstate the session on some error.

Parameters:

`[in]` *Session* The iSCSI session

Return values:

EFI_SUCCES	The session is reinstated from some error.
Other	Reinstatement failed.

Reinstate the session on some error.

Parameters:

`[in]` *Session* The iSCSI session

Return values:

EFI_SUCCESS	The session is reinstated from some error.
Other	Reinstatement failed.

References IScsiSessionAbort(), IScsiSessionInit(), IScsiSessionLogin(), and SESSION_STATE_FREE.

Referenced by IScsiExtScsiPassThruFunction().

EFI_STATUS IScsiSessionReLogin (IN ISCSI_SESSION * *Session*)

Wait for IPsec negotiation, then try to login the iSCSI session again.

Parameters:

`[in]` *Session* The iSCSI session

Return values:

EFI_SUCCESS	The iSCSI session login procedure finished.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_PROTOCOL_ERROR	Some kind of iSCSI protocol error happened.

Wait for IPsec negotiation, then try to login the iSCSI session again.

Parameters:

`[in]` *Session* The iSCSI session.

Return values:

EFI_SUCCESS	The iSCSI session login procedure finished.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.

EFI_PROTOCOL_ERROR	Some kind of iSCSI protocol error occurred.

References ISCSI_WAIT_IPSEC_TIMEOUT, and IScsiSessionLogin().

Referenced by IScsiStart().

```
EFI_STATUS IScsiUpdateTargetAddress ( IN OUT ISCSI_SESSION *   Session,
                                      IN CHAR8 *               Data,
                                      IN UINT32                Len
                                    )
```

Updated the target information according the data received in the iSCSI login response with an target redirection status.

Parameters:

`[in,out]`	*Session*	The iSCSI session.
`[in]`	*Data*	The data segment which should contain the TargetAddress key-value list.
`[in]`	*Len*	Length of the data.

Return values:

EFI_SUCCESS	The target address is updated.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_NOT_FOUND	The TargetAddress key is not found.
Others	Other errors as indicated.

Updated the target information according the data received in the iSCSI login response with an target redirection status.

Parameters:

`[in,out]`	*Session*	The iSCSI session.
`[in]`	*Data*	The data segment that should contain the TargetAddress key-value list.
`[in]`	*Len*	Length of the data.

Return values:

EFI_SUCCESS	The target address is updated.
EFI_OUT_OF_RESOURCES	Failed to allocate memory.
EFI_NOT_FOUND	The TargetAddress key is not found.
Others	Other errors as indicated.

References IP_MODE_AUTOCONFIG, IpMode, ISCSI_KEY_TARGET_ADDRESS, ISCSI_WELL_KNOWN_PORT, IScsiAsciiStrToIp(), IScsiBuildKeyValueList(), IScsiFreeKeyValueList(), and IScsiGetValueByKeyFromList().

Referenced by IScsiProcessLoginRsp().

NetworkPkg/Mtftp6Dxe/ComponentName.c File Reference

Functions

EFI_STATUS EFIAPI Mtftp6ComponentNameGetDriverName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN CHAR8 *Language, OUT CHAR16 **DriverName)

EFI_STATUS EFIAPI Mtftp6ComponentNameGetControllerName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE ChildHandle, IN CHAR8 *Language, OUT CHAR16 **ControllerName)

EFI_STATUS UpdateName (IN EFI_MTFTP6_PROTOCOL *Mtftp6)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gMtftp6ComponentName

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gMtftp6ComponentName2

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mMtftp6DriverNameTable []

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE * gMtftp6ControllerNameTable = NULL

Detailed Description

UEFI Component Name(2) protocol implementation for Mtftp6 driver.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI Mtftp6ComponentNameGetControllerName	(	IN EFI_COMPONENT_NAME_PROTOCOL *	*This*,
		IN EFI_HANDLE	*ControllerHandle*,
		IN EFI_HANDLE	*ChildHandle*,
		IN CHAR8 *	*Language*,

OUT CHAR16 ** *ControllerName*
)

Retrieves a Unicode string that is the user-readable name of the controller that is being managed by a driver.

This function retrieves the user-readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

`[in]`	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
`[in]`	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
`[in]`	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for bus drivers attempting to retrieve the name of the bus controller. It will not be NULL for a bus driver that attempts to retrieve the name of a child controller.
`[in]`	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
`[out]`	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL, and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

Retrieves a Unicode string that is the user-readable name of the controller that is being managed by a driver.

This function retrieves the user-readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
[in]	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers attempting to retrieve the name of the bus controller. It will not be NULL for a bus driver that attempts to retrieve the name of a child controller.
[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL, and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gMtftp6ComponentName, gMtftp6ControllerNameTable, and UpdateName().

```
EFI_STATUS EFIAPI
Mtftp6ComponentNameGetDriverName
  (
    IN EFI_COMPONENT_NAME_PROTOCOL *   This,
    IN CHAR8 *                         Language,
    OUT CHAR16 **                      DriverName
  )
```

Retrieves a Unicode string that is the user-readable name of the driver.

This function retrieves the user-readable name of a driver in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the driver name is returned in DriverName, and EFI_SUCCESS is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.

[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*DriverName*	A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by This in the language specified by Language.

Return values:

EFI_SUCCESS	The Unicode string for the Driver specified by This and the language specified by Language was returned in DriverName.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	DriverName is NULL.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gMtftp6ComponentName, and mMtftp6DriverNameTable.

EFI_STATUS UpdateName (IN EFI_MTFTP6_PROTOCOL * *Mtftp6*)

Update the component name for the Mtftp6 child handle.

Parameters:

Mtftp6[in] A pointer to the EFI_MTFTP6_PROTOCOL.

Return values:

EFI_SUCCESS	Update the ControllerNameTable of this instance successfully.
EFI_INVALID_PARAMETER	The input parameter is invalid.

References gMtftp6ComponentName, gMtftp6ComponentName2, and gMtftp6ControllerNameTable.

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gMtftp6ComponentName

Initial value:

```
{
 Mtftp6ComponentNameGetDriverName,
 Mtftp6ComponentNameGetControllerName,
 "eng"
}
```

Referenced by Mtftp6ComponentNameGetControllerName(), Mtftp6ComponentNameGetDriverName(), Mtftp6DriverEntryPoint(), and UpdateName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gMtftp6ComponentName2

Initial value:

```
{
 (EFI_COMPONENT_NAME2_GET_DRIVER_NAME) Mtftp6ComponentNameGetDriverName,
 (EFI_COMPONENT_NAME2_GET_CONTROLLER_NAME) Mtftp6ComponentNameGetControllerName,
 "en"
}
```

Referenced by Mtftp6DriverEntryPoint(), and UpdateName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE* gMtftp6ControllerNameTable = NULL

Referenced by Mtftp6ComponentNameGetControllerName(), and UpdateName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mMtftp6DriverNameTable[]

Initial value:

```
 {
  {
    "eng;en",
    L"MTFTP6 Network Service Driver"
  },
  {
    NULL,
    NULL
  }
}
```

Referenced by Mtftp6ComponentNameGetDriverName().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Mtftp6Dxe/Mtftp6Driver.c File Reference

Functions

VOID	Mtftp6DestroyService (IN MTFTP6_SERVICE *Service)
EFI_STATUS	Mtftp6CreateService (IN EFI_HANDLE Controller, IN EFI_HANDLE Image, OUT MTFTP6_SERVICE **Service)
VOID	Mtftp6DestroyInstance (IN MTFTP6_INSTANCE *Instance)
EFI_STATUS	Mtftp6CreateInstance (IN MTFTP6_SERVICE *Service, OUT MTFTP6_INSTANCE **Instance)
EFI_STATUS EFIAPI	Mtftp6DestroyChildEntryInHandleBuffer (IN LIST_ENTRY *Entry, IN VOID *Context)
EFI_STATUS EFIAPI	Mtftp6DriverEntryPoint (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)
EFI_STATUS EFIAPI	Mtftp6DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE Controller, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI	Mtftp6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE Controller, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI	Mtftp6DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE Controller, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)
EFI_STATUS EFIAPI	Mtftp6ServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN OUT EFI_HANDLE *ChildHandle)
EFI_STATUS EFIAPI	Mtftp6ServiceBindingDestroyChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE ChildHandle)

Variables

EFI_DRIVER_BINDING_PROTOCOL	gMtftp6DriverBinding
EFI_SERVICE_BINDING_PROTOCOL	gMtftp6ServiceBindingTemplate

Detailed Description

Driver Binding functions and Service Binding functions implementation for Mtftp6 Driver.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS Mtftp6CreateInstance ( IN MTFTP6_SERVICE *        Service,
                                  OUT MTFTP6_INSTANCE **     Instance
                                )
```

Create the MTFTP6 instance and initialize it.

Parameters:

[in]	*Service*	The pointer to the MTFTP6 service.
[out]	*Instance*	The pointer to the MTFTP6 instance.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate resources.
EFI_SUCCESS	The MTFTP6 instance is created.

References _MTFTP6_INSTANCE::BlkList, gMtftp6ProtocolTemplate, _MTFTP6_INSTANCE::InDestroy, _MTFTP6_INSTANCE::Link, _MTFTP6_INSTANCE::Mtftp6, MTFTP6_INSTANCE_SIGNATURE, _MTFTP6_INSTANCE::Service, and _MTFTP6_INSTANCE::Signature.

Referenced by Mtftp6ServiceBindingCreateChild().

```
EFI_STATUS Mtftp6CreateService ( IN EFI_HANDLE              Controller,
                                 IN EFI_HANDLE              Image,
                                 OUT MTFTP6_SERVICE **      Service
                               )
```

Create then initialize a MTFTP6 service binding instance.

Parameters:

[in]	*Controller*	The controller to install the MTFTP6 service binding on.
[in]	*Image*	The driver binding image of the MTFTP6 driver.
[out]	*Service*	The variable to receive the created service binding instance.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate resources to create the instance
EFI_DEVICE_ERROR	Failed to create a NULL UDP port to keep connection with UDP.
EFI_SUCCESS	The service instance is created for the controller.

References _MTFTP6_SERVICE::Children, _MTFTP6_SERVICE::ChildrenNum, _MTFTP6_SERVICE::Controller, _MTFTP6_SERVICE::DummyUdpIo, gMtftp6ServiceBindingTemplate, _MTFTP6_SERVICE::Image, MTFTP6_SERVICE_SIGNATURE, Mtftp6ConfigDummyUdpIo(), Mtftp6OnTimerTick(), _MTFTP6_SERVICE::ServiceBinding, _MTFTP6_SERVICE::Signature, and _MTFTP6_SERVICE::Timer.

Referenced by Mtftp6DriverBindingStart().

EFI_STATUS EFIAPI Mtftp6DestroyChildEntryInHandleBuffer (IN LIST_ENTRY * *Entry*,
IN VOID * *Context*
)

Callback function which provided by user to remove one node in NetDestroyLinkList process.

Parameters:

[in] *Entry* The entry to be removed.
[in] *Context* Pointer to the callback context corresponds to the Context in NetDestroyLinkList.

Return values:

EFI_SUCCESS The entry has been removed successfully.
Others Fail to remove the entry.

References _MTFTP6_INSTANCE::Handle, and MTFTP6_INSTANCE_SIGNATURE.

Referenced by Mtftp6DriverBindingStop().

VOID Mtftp6DestroyInstance (IN MTFTP6_INSTANCE * *Instance*)

Destroy the MTFTP6 instance and recycle the resources.

Parameters:

[in] *Instance* The pointer to the MTFTP6 instance.

Referenced by Mtftp6ServiceBindingCreateChild(), and Mtftp6ServiceBindingDestroyChild().

VOID Mtftp6DestroyService (IN MTFTP6_SERVICE * *Service*)

Destroy the MTFTP6 service. The MTFTP6 service may be partly initialized, or partly destroyed. If a resource is destroyed, it is marked as such in case the destroy failed and is called again later.

Parameters:

[in] *Service* The MTFTP6 service to be destroyed.

Referenced by Mtftp6DriverBindingStart(), and Mtftp6DriverBindingStop().

EFI_STATUS EFIAPI Mtftp6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL * *This*,
IN EFI_HANDLE *Controller*,
IN EFI_DEVICE_PATH_PROTOCOL * *RemainingDevicePath*
)

Start this driver on Controller. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Start() it must also follow these calling restrictions.

Parameters:

[in] *This* Protocol instance pointer.
[in] *Controller* Handle of device to bind driver to.
[in] *RemainingDevicePath* Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	This driver is added to Controller.
EFI_ALREADY_STARTED	This driver is already running on Controller.
Others	This driver does not support this device.

References Mtftp6CreateService(), Mtftp6DestroyService(), _MTFTP6_SERVICE::ServiceBinding, and _MTFTP6_SERVICE::Timer.

```
EFI_STATUS EFIAPI
Mtftp6DriverBindingStop (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     Controller,
  IN UINTN                          NumberOfChildren,
  IN EFI_HANDLE *                   ChildHandleBuffer
)
```

Stop this driver on Controller. This service is called by the EFI boot service DisconnectController(). In order to make drivers as small as possible, there are calling restrictions for this service. DisconnectController() must follow these calling restrictions. If any other agent wishes to call Stop(), it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*Controller*	Handle of device to stop driver on
[in]	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If number of children is zero, stop the entire bus driver.
[in]	*ChildHandleBuffer*	List of Child Handles to Stop.

Return values:

EFI_SUCCESS	This driver is removed Controller.
EFI_DEVICE_ERROR	An unexpected error.
Others	This driver was not removed from this device.

References MTFTP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer, _MTFTP6_SERVICE::Children, MTFTP6_SERVICE_FROM_THIS, Mtftp6DestroyChildEntryInHandleBuffer(), Mtftp6DestroyService(), MTFTP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren, and MTFTP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding.

```
EFI_STATUS EFIAPI
Mtftp6DriverBindingSupported (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     Controller,
  IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
)
```

Test to see if this driver supports Controller. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Supported(), it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.

[in]	*Controller*	Handle of device to test
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child. device to start.

Return values:

EFI_SUCCESS	This driver supports this device.
Others	This driver does not support this device.

```
EFI_STATUS EFIAPI Mtftp6DriverEntryPoint ( IN EFI_HANDLE          ImageHandle,
                                           IN EFI_SYSTEM_TABLE *  SystemTable
                                         )
```

This is the declaration of an EFI image entry point. This entry point is the same for UEFI Applications, UEFI OS Loaders, and UEFI Drivers, including both device drivers and bus drivers.

Entry point of the MTFTP6 driver to install various protocols.

Parameters:

[in]	*ImageHandle*	The firmware allocated handle for the UEFI image.
[in]	*SystemTable*	The pointer to the EFI System Table.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.

References gMtftp6ComponentName, gMtftp6ComponentName2, and gMtftp6DriverBinding.

```
EFI_STATUS EFIAPI                    ( IN EFI_SERVICE_BINDING_PROTOCOL *  This,
Mtftp6ServiceBindingCreateChild        IN OUT EFI_HANDLE *                ChildHandle
                                     )
```

Creates a child handle and installs a protocol.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

[in]	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
[in,out]	*ChildHandle*	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.
EFI_INVALID_PARAMETER	ChildHandle is NULL.
Others	The child handle was not created.

References _MTFTP6_SERVICE::Children, _MTFTP6_SERVICE::ChildrenNum, _MTFTP6_SERVICE::DummyUdpIo, gMtftp6DriverBinding, _MTFTP6_INSTANCE::Handle, _MTFTP6_INSTANCE::Link, _MTFTP6_INSTANCE::Mtftp6, MTFTP6_SERVICE_FROM_THIS, Mtftp6CreateInstance(), and Mtftp6DestroyInstance().

EFI_STATUS EFIAPI Mtftp6ServiceBindingDestroyChild	(	IN EFI_SERVICE_BINDING_PROTOCOL *	*This*,
		IN EFI_HANDLE	*ChildHandle*
	)		

Destroys a child handle with a protocol installed on it.

The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

[in]	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
[in]	*ChildHandle*	Handle of the child to destroy.

Return values:

EFI_SUCCES	The protocol was removed from ChildHandle.
EFI_UNSUPPORTED	ChildHandle does not support the protocol that is being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
Others	The child handle was not destroyed

References _MTFTP6_SERVICE::ChildrenNum, _MTFTP6_SERVICE::DummyUdpIo, gMtftp6DriverBinding, _MTFTP6_INSTANCE::Handle, _MTFTP6_INSTANCE::InDestroy, _MTFTP6_INSTANCE::Link, _MTFTP6_INSTANCE::McastUdpIo, MTFTP6_INSTANCE_FROM_THIS, MTFTP6_SERVICE_FROM_THIS, Mtftp6DestroyInstance(), _MTFTP6_INSTANCE::Service, and _MTFTP6_INSTANCE::UdpIo.

Variable Documentation

EFI_DRIVER_BINDING_PROTOCOL gMtftp6DriverBinding

Initial value:

```
{
  Mtftp6DriverBindingSupported,
  Mtftp6DriverBindingStart,
  Mtftp6DriverBindingStop,
  0xa,
  NULL,
  NULL
}
```

Referenced by Mtftp6DriverEntryPoint(), Mtftp6ServiceBindingCreateChild(), and Mtftp6ServiceBindingDestroyChild().

EFI_SERVICE_BINDING_PROTOCOL gMtftp6ServiceBindingTemplate

Initial value:

```
{
  Mtftp6ServiceBindingCreateChild,
  Mtftp6ServiceBindingDestroyChild
}
```

Referenced by Mtftp6CreateService().

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Mtftp6Dxe/Mtftp6Driver.h File Reference

Functions

EFI_STATUS EFIAPI Mtftp6DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE Controller, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI Mtftp6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE Controller, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI Mtftp6DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE Controller, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)
EFI_STATUS EFIAPI Mtftp6ServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN OUT EFI_HANDLE *ChildHandle)
EFI_STATUS EFIAPI Mtftp6ServiceBindingDestroyChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE ChildHandle)

Variables

EFI_COMPONENT_NAME_PROTOCOL gMtftp6ComponentName
EFI_COMPONENT_NAME2_PROTOCOL gMtftp6ComponentName2
EFI_UNICODE_STRING_TABLE * gMtftp6ControllerNameTable

Detailed Description

Driver Binding functions and Service Binding functions declaration for Mtftp6 Driver.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI Mtftp6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL * *This*,

```
                                IN EFI_HANDLE                    Controller,
                                IN EFI_DEVICE_PATH_PROTOCOL *    RemainingDevicePath
                              )
```

Start this driver on Controller. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Start() it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*Controller*	Handle of device to bind driver to.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	This driver is added to Controller.
EFI_ALREADY_STARTED	This driver is already running on Controller.
Others	This driver does not support this device.

References Mtftp6CreateService(), Mtftp6DestroyService(), _MTFTP6_SERVICE::ServiceBinding, and _MTFTP6_SERVICE::Timer.

```
EFI_STATUS EFIAPI
Mtftp6DriverBindingStop      ( IN EFI_DRIVER_BINDING_PROTOCOL *   This,
                               IN EFI_HANDLE                      Controller,
                               IN UINTN                           NumberOfChildren,
                               IN EFI_HANDLE *                    ChildHandleBuffer
                             )
```

Stop this driver on Controller. This service is called by the EFI boot service DisconnectController(). In order to make drivers as small as possible, there are calling restrictions for this service. DisconnectController() must follow these calling restrictions. If any other agent wishes to call Stop(), it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*Controller*	Handle of device to stop driver on
[in]	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If number of children is zero, stop the entire bus driver.
[in]	*ChildHandleBuffer*	List of Child Handles to Stop.

Return values:

EFI_SUCCESS	This driver is removed Controller.
EFI_DEVICE_ERROR	An unexpected error.
Others	This driver was not removed from this device.

References MTFTP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer, _MTFTP6_SERVICE::Children, MTFTP6_SERVICE_FROM_THIS, Mtftp6DestroyChildEntryInHandleBuffer(), Mtftp6DestroyService(), MTFTP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren, and MTFTP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding.

```
EFI_STATUS EFIAPI
Mtftp6DriverBindingSupported
( IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     Controller,
  IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
)
```

Test to see if this driver supports Controller. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Supported(), it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*Controller*	Handle of device to test.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	This driver supports this device.
Others	This driver does not support this device.

Test to see if this driver supports Controller. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Supported(), it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*Controller*	Handle of device to test
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child. device to start.

Return values:

EFI_SUCCESS	This driver supports this device.
Others	This driver does not support this device.

```
EFI_STATUS EFIAPI
Mtftp6ServiceBindingCreateChild
( IN EFI_SERVICE_BINDING_PROTOCOL *  This,
  IN OUT EFI_HANDLE *                ChildHandle
)
```

Creates a child handle and installs a protocol.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

[in]	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
[in,out]	*ChildHandle*	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.
EFI_INVALID_PARAMETER	ChildHandle is NULL.
Others	The child handle was not created.

References _MTFTP6_SERVICE::Children, _MTFTP6_SERVICE::ChildrenNum, _MTFTP6_SERVICE::DummyUdpIo, gMtftp6DriverBinding, _MTFTP6_INSTANCE::Handle, _MTFTP6_INSTANCE::Link, _MTFTP6_INSTANCE::Mtftp6, MTFTP6_SERVICE_FROM_THIS, Mtftp6CreateInstance(), and Mtftp6DestroyInstance().

```
EFI_STATUS EFIAPI
Mtftp6ServiceBindingDestroyChild (
  IN EFI_SERVICE_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                      ChildHandle
)
```

Destroys a child handle with a protocol installed on it.

The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

[in]	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
[in]	*ChildHandle*	Handle of the child to destroy.

Return values:

EFI_SUCCES	The protocol was removed from ChildHandle.
EFI_UNSUPPORTED	ChildHandle does not support the protocol that is being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
Others	The child handle was not destroyed

References _MTFTP6_SERVICE::ChildrenNum, _MTFTP6_SERVICE::DummyUdpIo, gMtftp6DriverBinding, _MTFTP6_INSTANCE::Handle, _MTFTP6_INSTANCE::InDestroy, _MTFTP6_INSTANCE::Link, _MTFTP6_INSTANCE::McastUdpIo, MTFTP6_INSTANCE_FROM_THIS, MTFTP6_SERVICE_FROM_THIS, Mtftp6DestroyInstance(), _MTFTP6_INSTANCE::Service, and _MTFTP6_INSTANCE::UdpIo.

Variable Documentation

EFI_COMPONENT_NAME_PROTOCOL gMtftp6ComponentName

Referenced by Mtftp6ComponentNameGetControllerName(), Mtftp6ComponentNameGetDriverName(), Mtftp6DriverEntryPoint(), and UpdateName().

EFI_COMPONENT_NAME2_PROTOCOL gMtftp6ComponentName2

Referenced by Mtftp6DriverEntryPoint(), and UpdateName().

EFI_UNICODE_STRING_TABLE* gMtftp6ControllerNameTable

Referenced by Mtftp6ComponentNameGetControllerName(), and UpdateName().

Generated on Mon Sep 28 08:49:56 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Mtftp6Dxe/Mtftp6Impl.c File Reference

Functions

EFI_STATUS EFIAPI EfiMtftp6GetModeData (IN EFI_MTFTP6_PROTOCOL *This, OUT EFI_MTFTP6_MODE_DATA *ModeData)
EFI_STATUS EFIAPI EfiMtftp6Configure (IN EFI_MTFTP6_PROTOCOL *This, IN EFI_MTFTP6_CONFIG_DATA *MtftpConfigData)
EFI_STATUS EFIAPI EfiMtftp6GetInfo (IN EFI_MTFTP6_PROTOCOL *This, IN EFI_MTFTP6_OVERRIDE_DATA *OverrideData, IN UINT8 *Filename, IN UINT8 *ModeStr, IN UINT8 OptionCount, IN EFI_MTFTP6_OPTION *OptionList, OUT UINT32 *PacketLength, OUT EFI_MTFTP6_PACKET **Packet)
EFI_STATUS EFIAPI EfiMtftp6ParseOptions (IN EFI_MTFTP6_PROTOCOL *This, IN UINT32 PacketLen, IN EFI_MTFTP6_PACKET *Packet, OUT UINT32 *OptionCount, OUT EFI_MTFTP6_OPTION **OptionList)
EFI_STATUS EFIAPI EfiMtftp6ReadFile (IN EFI_MTFTP6_PROTOCOL *This, IN EFI_MTFTP6_TOKEN *Token)
EFI_STATUS EFIAPI EfiMtftp6WriteFile (IN EFI_MTFTP6_PROTOCOL *This, IN EFI_MTFTP6_TOKEN *Token)
EFI_STATUS EFIAPI EfiMtftp6ReadDirectory (IN EFI_MTFTP6_PROTOCOL *This, IN EFI_MTFTP6_TOKEN *Token)
EFI_STATUS EFIAPI EfiMtftp6Poll (IN EFI_MTFTP6_PROTOCOL *This)

Variables

EFI_MTFTP6_PROTOCOL gMtftp6ProtocolTemplate

Detailed Description

This EFI_MTFTP6_PROTOCOL interface implementation.

It supports the following RFCs: RFC1350 - THE TFTP PROTOCOL (REVISION 2) RFC2090 - TFTP Multicast Option RFC2347 - TFTP Option Extension RFC2348 - TFTP Blocksize Option RFC2349 - TFTP Timeout Interval and Transfer Size Options

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI EfiMtftp6Configure (IN EFI_MTFTP6_PROTOCOL * *This*,

```
                                    IN EFI_MTFTP6_CONFIG_DATA *   MtftpConfigData
                                  )
```

Initializes, changes, or resets the default operational setting for this EFI MTFTPv6 Protocol driver instance.

The Configure() function is used to set and change the configuration data for this EFI MTFTPv6 Protocol driver instance. The configuration data can be reset to startup defaults by calling Configure() with MtftpConfigData set to NULL. Whenever the instance is reset, any pending operation is aborted. By changing the EFI MTFTPv6 Protocol driver instance configuration data, the client can connect to different MTFTPv6 servers. The configuration parameters in MtftpConfigData are used as the default parameters in later MTFTPv6 operations and can be overridden in later operations.

Parameters:

[in]	*This*	Pointer to the EFI_MTFTP6_PROTOCOL instance.
[in]	*MtftpConfigData*	Pointer to the configuration data structure.

Return values:

EFI_SUCCESS	The EFI MTFTPv6 Protocol instance was configured successfully.
EFI_INVALID_PARAMETER	One or more following conditions are TRUE: ◊ This is NULL. ◊ MtftpConfigData.StationIp is neither zero nor one of the configured IP addresses in the underlying IPv6 driver. ◊ MtftpCofigData.ServerIp is not a valid IPv6 unicast address. Note: It does not match the UEFI 2.3 Specification.
EFI_ACCESS_DENIED	- The configuration could not be changed at this time because there is some MTFTP background operation in progress. ◊ MtftpCofigData.LocalPort is already in use. Note: It does not match the UEFI 2.3 Specification.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_OUT_OF_RESOURCES	The EFI MTFTPv6 Protocol driver instance data could not be allocated. Note: It is not defined in the UEFI 2.3 Specification.
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI MTFTPv6 Protocol driver instance is not configured. Note: It is not defined in the UEFI 2.3 Specification.

References _MTFTP6_INSTANCE::Config, _MTFTP6_SERVICE::Controller, _MTFTP6_INSTANCE::Handle, _MTFTP6_SERVICE::Image, MTFTP6_INSTANCE_FROM_THIS, Mtftp6ConfigDummyUdpIo(), Mtftp6OperationClean(), _MTFTP6_INSTANCE::Service, and _MTFTP6_INSTANCE::UdpIo.

```
EFI_STATUS EFIAPI EfiMtftp6GetInfo  ( IN EFI_MTFTP6_PROTOCOL *          This,
                                      IN EFI_MTFTP6_OVERRIDE_DATA *     OverrideData,
                                      IN UINT8 *                        Filename,
                                      IN UINT8 *                        ModeStr,
                                      IN UINT8                          OptionCount,
                                      IN EFI_MTFTP6_OPTION *            OptionList,
                                      OUT UINT32 *                      PacketLength,
                                      OUT EFI_MTFTP6_PACKET **          Packet
                                    )
```

Get the information of the download from the server.

The GetInfo() function assembles an MTFTPv6 request packet with options, sends it to the MTFTPv6 server, and may return an MTFTPv6 OACK, MTFTPv6 ERROR, or ICMP ERROR packet. Retries occur only if no response packets are received from the MTFTPv6 server before the timeout expires.

Parameters:

[in]	*This*	Pointer to the EFI_MTFTP6_PROTOCOL instance.
[in]	*OverrideData*	Data that is used to override the existing parameters. If NULL, the default parameters that were set in the EFI_MTFTP6_PROTOCOL.Configure() function are used.
[in]	*Filename*	Pointer to null-terminated ASCII file name string.
[in]	*ModeStr*	Pointer to null-terminated ASCII mode string. If NULL, octet will be used.
[in]	*OptionCount*	Number of option/value string pairs in OptionList.
[in]	*OptionList*	Pointer to array of option/value string pairs. Ignored if OptionCount is zero.
[out]	*PacketLength*	The number of bytes in the returned packet.
[out]	*Packet*	The pointer to the received packet. This buffer must be freed by the caller.

Return values:

EFI_SUCCESS	An MTFTPv6 OACK packet was received and is in the Packet. Note: It does not match the UEFI 2.3 Specification.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Filename is NULL. ◊ OptionCount is not zero and OptionList is NULL. ◊ One or more options in OptionList have wrong format. ◊ PacketLength is NULL. ◊ OverrideData.ServerIp is not valid unicast IPv6 addresses.
EFI_UNSUPPORTED	One or more options in the OptionList are unsupported by this implementation.
EFI_NOT_STARTED	The EFI MTFTPv6 Protocol driver has not been started.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_ACCESS_DENIED	The previous operation has not completed yet.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_TFTP_ERROR	An MTFTPv6 ERROR packet was received and is in the Packet.
EFI_NETWORK_UNREACHABLE	An ICMP network unreachable error packet was received and the Packet is set to NULL. Note: It is not defined in UEFI 2.3 Specification.
EFI_HOST_UNREACHABLE	An ICMP host unreachable error packet was received and the Packet is set to NULL. Note: It is not defined in the UEFI 2.3 Specification.
EFI_PROTOCOL_UNREACHABLE	An ICMP protocol unreachable error packet was received and the Packet is set to NULL. Note: It is not defined in the UEFI 2.3 Specification.
EFI_PORT_UNREACHABLE	An ICMP port unreachable error packet was received and the Packet is set to NULL.
EFI_ICMP_ERROR	Some other ICMP ERROR packet was received and the Packet is set to NULL. Note: It does not match the UEFI 2.3 Specification.
EFI_PROTOCOL_ERROR	

	An unexpected MTFTPv6 packet was received and is in the Packet.
EFI_TIMEOUT	No responses were received from the MTFTPv6 server.
EFI_DEVICE_ERROR	An unexpected network error or system error occurred.
EFI_NO_MEDIA	There was a media error.

References Mtftp6CheckPacket(), Mtftp6OperationStart(), _MTFTP6_GETINFO_CONTEXT::Packet, _MTFTP6_GETINFO_CONTEXT::PacketLen, and _MTFTP6_GETINFO_CONTEXT::Status.

```
EFI_STATUS EFIAPI EfiMtftp6GetModeData ( IN EFI_MTFTP6_PROTOCOL *      This,
                                         OUT EFI_MTFTP6_MODE_DATA *    ModeData
                                       )
```

Returns the current operating mode data for the MTFTP6 instance.

The GetModeData() function returns the current operating mode and cached data packet for the MTFTP6 instance.

Parameters:

[in]	*This*	Pointer to the EFI_MTFTP6_PROTOCOL instance.
[out]	*ModeData*	The buffer in which the EFI MTFTPv6 Protocol driver mode data is returned.

Return values:

EFI_SUCCESS	The configuration data was returned successfully.
EFI_OUT_OF_RESOURCES	The required mode data could not be allocated.
EFI_INVALID_PARAMETER	This is NULL or ModeData is NULL.

References _MTFTP6_INSTANCE::Config, mMtftp6SupportedOptions, MTFTP6_INSTANCE_FROM_THIS, and MTFTP6_SUPPORTED_OPTIONS_NUM.

```
EFI_STATUS EFIAPI EfiMtftp6ParseOptions ( IN EFI_MTFTP6_PROTOCOL *   This,
                                          IN UINT32                  PacketLen,
                                          IN EFI_MTFTP6_PACKET *     Packet,
                                          OUT UINT32 *               OptionCount,
                                          OUT EFI_MTFTP6_OPTION **   OptionList
                                        )
```

Parse the options in an MTFTPv6 OACK packet.

The ParseOptions() function parses the option fields in an MTFTPv6 OACK packet and returns the number of options that were found, and optionally, a list of pointers to the options in the packet. If one or more of the option fields are not valid, then EFI_PROTOCOL_ERROR is returned and OptionCount and *OptionList stop at the last valid option.

Parameters:

[in]	*This*	Pointer to the EFI_MTFTP6_PROTOCOL instance.
[in]	*PacketLen*	Length of the OACK packet to be parsed.
[in]	*Packet*	Pointer to the OACK packet to be parsed.
[out]	*OptionCount*	Pointer to the number of options in the following OptionList.
[out]	*OptionList*	Pointer to EFI_MTFTP6_OPTION storage. Each pointer in the OptionList points to the corresponding MTFTP option buffer in the Packet. Call the EFI

Boot Service FreePool() to release the OptionList if the options in this OptionList are not needed anymore.

Return values:

EFI_SUCCESS	The OACK packet was valid and the OptionCount and OptionList parameters have been updated.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ PacketLen is 0. ◊ Packet is NULL or Packet is not a valid MTFTPv6 packet. ◊ OptionCount is NULL.
EFI_NOT_FOUND	No options were found in the OACK packet.
EFI_OUT_OF_RESOURCES	Storage for the OptionList array can not be allocated.
EFI_PROTOCOL_ERROR	One or more of the option fields is invalid.

References Mtftp6ParseStart().

EFI_STATUS EFIAPI EfiMtftp6Poll (IN EFI_MTFTP6_PROTOCOL * *This*)

Polls for incoming data packets and processes outgoing data packets.

The Poll() function can be used by network drivers and applications to increase the rate that data packets are moved between the communications device and the transmit and receive queues. In some systems, the periodic timer event in the managed network driver may not poll the underlying communications device fast enough to transmit and/or receive all data packets without missing incoming packets or dropping outgoing packets. Drivers and applications that are experiencing packet loss should try calling the Poll() function more often.

Parameters:

`[in]` *This* The MTFTP6 protocol instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_NOT_STARTED	This EFI MTFTPv6 Protocol instance has not been started.
EFI_INVALID_PARAMETER	This is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_TIMEOUT	Data was dropped out of the transmit and/or receive queue. Consider increasing the polling rate.

References _MTFTP6_INSTANCE::Config, MTFTP6_INSTANCE_FROM_THIS, and _MTFTP6_INSTANCE::UdpIo.

EFI_STATUS EFIAPI EfiMtftp6ReadDirectory (IN EFI_MTFTP6_PROTOCOL * *This*,
IN EFI_MTFTP6_TOKEN * *Token*
)

Download a data file directory from an MTFTPv6 server.

The ReadDirectory() function is used to return a list of files on the MTFTPv6 server that are logically (or operationally) related to Token.Filename. The directory request packet that is sent to the server is built with the option list that was provided by the caller, if present. The file information that the server returns is put into either of the following locations:

- A fixed buffer that is pointed to by Token.Buffer.

- A download service function that is pointed to by Token.CheckPacket. If both Token.Buffer and Token.CheckPacket are used, then Token.CheckPacket will be called first. If the call is successful, the packet will be stored in Token.Buffer.

Parameters:

[in] *This* Pointer to the EFI_MTFTP6_PROTOCOL instance.

[in] *Token* Pointer to the token structure to provide the parameters that are used in this operation.

Return values:

EFI_SUCCESS	The MTFTPv6 related file "directory" has been downloaded.
EFI_UNSUPPORTED	The EFI MTFTPv6 Protocol driver does not support this function.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token.Filename is NULL. ◊ Token.OptionCount is not zero and Token.OptionList is NULL. ◊ One or more options in Token.OptionList have wrong format. ◊ Token.Buffer and Token.CheckPacket are both NULL. ◊ Token.OverrideData.ServerIp is not valid unicast IPv6 addresses.
EFI_UNSUPPORTED	One or more options in the Token.OptionList are not supported by this implementation.
EFI_NOT_STARTED	The EFI MTFTPv6 Protocol driver has not been started.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_ALREADY_STARTED	This Token is already being used in another MTFTPv6 session.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_ACCESS_DENIED	The previous operation has not completed yet.
EFI_DEVICE_ERROR	An unexpected network error or system error occurred.

References Mtftp6OperationStart().

```
EFI_STATUS EFIAPI EfiMtftp6ReadFile ( IN EFI_MTFTP6_PROTOCOL *  This,
                                      IN EFI_MTFTP6_TOKEN *     Token
                                    )
```

Download a file from an MTFTPv6 server.

The ReadFile() function is used to initialize and start an MTFTPv6 download process, and optionally, wait for completion. When the download operation completes, whether successfully or not, the Token.Status field is updated by the EFI MTFTPv6 Protocol driver, and then Token.Event is signaled if it is not NULL. Data can be downloaded from the MTFTPv6 server into either of the following locations:

- A fixed buffer that is pointed to by Token.Buffer
- A download service function that is pointed to by Token.CheckPacket. If both Token.Buffer and Token.CheckPacket are used, then Token.CheckPacket will be called first. If the call is successful, the packet will be stored in Token.Buffer.

Parameters:

[in] *This* Pointer to the EFI_MTFTP6_PROTOCOL instance.

[in] *Token* Pointer to the token structure to provide the parameters that are used in this operation.

Return values:

EFI_SUCCESS	The data file has been transferred successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_BUFFER_TOO_SMALL	BufferSize is not zero but not large enough to hold the downloaded data in downloading process. Note: It does not match the UEFI 2.3 Specification.
EFI_ABORTED	Current operation is aborted by user.
EFI_NETWORK_UNREACHABLE	An ICMP network unreachable error packet was received. Note: It is not defined in the UEFI 2.3 Specification.
EFI_HOST_UNREACHABLE	An ICMP host unreachable error packet was received. Note: It is not defined in the UEFI 2.3 Specification.
EFI_PROTOCOL_UNREACHABLE	An ICMP protocol unreachable error packet was received. Note: It is not defined in the UEFI 2.3 Specification.
EFI_PORT_UNREACHABLE	An ICMP port unreachable error packet was received. Note: It is not defined in the UEFI 2.3 Specification.
EFI_ICMP_ERROR	An ICMP ERROR packet was received.
EFI_TIMEOUT	No responses were received from the MTFTPv6 server.
EFI_TFTP_ERROR	An MTFTPv6 ERROR packet was received.
EFI_DEVICE_ERROR	An unexpected network error or system error occurred.
EFI_NO_MEDIA	There was a media error.

References MtftpOperationStart().

```
EFI_STATUS EFIAPI EfiMtftp6WriteFile ( IN EFI_MTFTP6_PROTOCOL *  This,
                                       IN EFI_MTFTP6_TOKEN *     Token
                                     )
```

Send a file to an MTFTPv6 server.

The WriteFile() function is used to initialize an uploading operation with the given option list and optionally wait for completion. If one or more of the options is not supported by the server, the unsupported options are ignored and a standard TFTP process starts instead. When the upload process completes, whether successfully or not, Token.Event is signaled, and the EFI MTFTPv6 Protocol driver updates Token.Status. The caller can supply the data to be uploaded in the following two modes:

- Through the user-provided buffer
- Through a callback function With the user-provided buffer, the Token.BufferSize field indicates the length of the buffer, and the driver will upload the data in the buffer. With an EFI_MTFTP6_PACKET_NEEDED callback function, the driver will call this callback function to get more data from the user to upload.

Parameters:

[in]	*This*	Pointer to the EFI_MTFTP6_PROTOCOL instance.
[in]	*Token*	Pointer to the token structure to provide the parameters that are used in this operation.

Return values:

EFI_SUCCESS	The upload session has started.
EFI_UNSUPPORTED	The operation is not supported by this implementation.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token.Filename is NULL.

◊ Token.OptionCount is not zero and Token.OptionList is NULL.
◊ One or more options in Token.OptionList have wrong format.
◊ Token.Buffer and Token.PacketNeeded are both NULL.
◊ Token.OverrideData.ServerIp is not a valid unicast IPv6 address.

EFI_UNSUPPORTED	One or more options in the Token.OptionList are not supported by this implementation.
EFI_NOT_STARTED	The EFI MTFTPv6 Protocol driver has not been started.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_ALREADY_STARTED	This Token is already being used in another MTFTPv6 session.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_ACCESS_DENIED	The previous operation has not completed yet.
EFI_DEVICE_ERROR	An unexpected network error or system error occurred.

References Mtftp6OperationStart().

Variable Documentation

EFI_MTFTP6_PROTOCOL gMtftp6ProtocolTemplate

Initial value:

```
{
 EfiMtftp6GetModeData,
 EfiMtftp6Configure,
 EfiMtftp6GetInfo,
 EfiMtftp6ParseOptions,
 EfiMtftp6ReadFile,
 EfiMtftp6WriteFile,
 EfiMtftp6ReadDirectory,
 EfiMtftp6Poll
 }
```

Referenced by Mtftp6CreateInstance().

NetworkPkg/Mtftp6Dxe/Mtftp6Impl.h File Reference

Data Structures

struct _MTFTP
struct _MTFTP
struct _MTFTP
struct MTFTP6

Defines

"memItemRight" valign="bottom">MTFTP6_SERVICE_SIGNATURE SIGNATURE_32 ('M', 'F', '6', 'S')
#define MTFTP6
'F', '6', 'I'
#define MTFTP6
#define MTFTP6
#define MTFTP6
#define MTFTP6
#define MTFTP6
#define MTFTP6
#define MTFTP6
MTFTP6
MTFTP6
#define MTFTP6
MTFTP6
MTFTP6

Typedefs

typedef struct _MTFTP6_SERVICE MTFTP6
typedef struct _MTFTP6_INSTANCE MTFTP6
typedef struct
_MTFTP6_GETINFO_CONTEXT MTFTP6

Functions

EFI_STATUS EFIAPI EfiMtftp
OUT EFI
EFI_STATUS EFIAPI EfiMtftp
EFI_MT
EFI_STATUS EFIAPI EfiMtftp
EFI_MT
*Filenam
EFI_MT
*PacketL
EFI_STATUS EFIAPI EfiMtftp
IN UINT
OUT UI
**Option
EFI_STATUS EFIAPI EfiMtftp

EFI_MTFTP6_T
EFI_STATUS EFIAPI EfiMtftp6WriteF
EFI_MTFTP6_T
EFI_STATUS EFIAPI EfiMtftp6ReadD
IN EFI_MTFTP6
EFI_STATUS EFIAPI EfiMtftp6Poll (IN

Variables

EFI_MTFTP6_PROTOCOL gMtftp6ProtocolT

Detailed Description

Mtftp6 internal data structure and definition declaration.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Mtftp6OperationStart().

#define MTFTP6_DEFAULT_MAX_RETRY 5

Referenced by Mtftp6OperationStart().

#define MTFTP6_DEFAULT_SERVER_CMD_PORT 69

Referenced by Mtftp6OperationStart().

#define MTFTP6_DEFAULT_TIMEOUT 3

Referenced by Mtftp6OperationStart().

#define MTFTP6_GET_MAPPING_TIMEOUT 3

Referenced by Mtftp6GetMapping().

#define MTFTP6_INSTANCE_FROM_THIS (a) CR (a, MTFTP6_INSTANCE, Mtftp6, MTFTP6_INSTANCE_SIGNATURE)

Referenced by EfiMtftp6Configure(), EfiMtftp6GetModeData(), EfiMtftp6Poll(), Mtftp6OperationStart(), and Mtftp6ServiceBindingDestroyChild().

#define MTFTP6_INSTANCE_SIGNATURE SIGNATURE_32 ('M', 'F', '6', 'I')

Referenced by Mtftp6CreateInstance(), Mtftp6DestroyChildEntryInHandleBuffer(), Mtftp6RrqInput(), and Mtftp6WrqInput().

#define MTFTP6_SERVICE_FROM_THIS (a) CR (a, MTFTP6_SERVICE, ServiceBinding, MTFTP6_SERVICE_SIGNATURE)

Referenced by Mtftp6DriverBindingStop(), Mtftp6ServiceBindingCreateChild(), and Mtftp6ServiceBindingDestroyChild().

#define MTFTP6_SERVICE_SIGNATURE SIGNATURE_32 ('M', 'F', '6', 'S')

Referenced by Mtftp6CreateService().

#define MTFTP6_TICK_PER_SECOND 10000000U

Referenced by Mtftp6GetMapping().

Typedef Documentation

typedef struct _MTFTP6_GETINFO_CONTEXT MTFTP6_GETINFO_CONTEXT

typedef struct _MTFTP6_INSTANCE MTFTP6_INSTANCE

typedef struct _MTFTP6_SERVICE MTFTP6_SERVICE

Function Documentation

EFI_STATUS EFIAPI EfiMtftp6Configure (IN EFI_MTFTP6_PROTOCOL * *This*,
IN EFI_MTFTP6_CONFIG_DATA * *MtftpConfigData*
)

Initializes, changes, or resets the default operational setting for this EFI MTFTPv6 Protocol driver instance.

The Configure() function is used to set and change the configuration data for this EFI MTFTPv6 Protocol driver instance. The configuration data can be reset to startup defaults by calling Configure() with MtftpConfigData set to NULL. Whenever the instance is reset, any pending operation is aborted. By changing

the EFI MTFTPv6 Protocol driver instance configuration data, the client can connect to different MTFTPv6 servers. The configuration parameters in MtftpConfigData are used as the default parameters in later MTFTPv6 operations and can be overridden in later operations.

Parameters:

[in]	*This*	Pointer to the EFI_MTFTP6_PROTOCOL instance.
[in]	*MtftpConfigData*	Pointer to the configuration data structure.

Return values:

EFI_SUCCESS	The EFI MTFTPv6 Protocol instance was configured successfully.
EFI_INVALID_PARAMETER	One or more following conditions are TRUE: ◊ This is NULL. ◊ MtftpConfigData.StationIp is neither zero nor one of the configured IP addresses in the underlying IPv6 driver. ◊ MtftpCofigData.ServerIp is not a valid IPv6 unicast address. Note: It does not match the UEFI 2.3 Specification.
EFI_ACCESS_DENIED	- The configuration could not be changed at this time because there is some MTFTP background operation in progress. ◊ MtftpCofigData.LocalPort is already in use. Note: It does not match the UEFI 2.3 Specification.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_OUT_OF_RESOURCES	The EFI MTFTPv6 Protocol driver instance data could not be allocated. Note: It is not defined in the UEFI 2.3 Specification.
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI MTFTPv6 Protocol driver instance is not configured. Note: It is not defined in the UEFI 2.3 Specification.

References _MTFTP6_INSTANCE::Config, _MTFTP6_SERVICE::Controller, _MTFTP6_INSTANCE::Handle, _MTFTP6_SERVICE::Image, MTFTP6_INSTANCE_FROM_THIS, Mtftp6ConfigDummyUdpIo(), Mtftp6OperationClean(), _MTFTP6_INSTANCE::Service, and _MTFTP6_INSTANCE::UdpIo.

```
EFI_STATUS EFIAPI EfiMtftp6GetInfo ( IN EFI_MTFTP6_PROTOCOL *          This,
                                     IN EFI_MTFTP6_OVERRIDE_DATA *     OverrideData,
                                     IN UINT8 *                        Filename,
                                     IN UINT8 *                        ModeStr,
                                     IN UINT8                          OptionCount,
                                     IN EFI_MTFTP6_OPTION *            OptionList,
                                     OUT UINT32 *                      PacketLength,
                                     OUT EFI_MTFTP6_PACKET **          Packet
                                   )
```

Get the information of the download from the server.

The GetInfo() function assembles an MTFTPv6 request packet with options, sends it to the MTFTPv6 server, and may return an MTFTPv6 OACK, MTFTPv6 ERROR, or ICMP ERROR packet. Retries occur only if no response packets are received from the MTFTPv6 server before the timeout expires.

Parameters:

[in]	*This*	Pointer to the EFI_MTFTP6_PROTOCOL instance.
[in]	*OverrideData*	

		Data that is used to override the existing parameters. If NULL, the default parameters that were set in the EFI_MTFTP6_PROTOCOL.Configure() function are used.
[in]	*Filename*	Pointer to null-terminated ASCII file name string.
[in]	*ModeStr*	Pointer to null-terminated ASCII mode string. If NULL, octet will be used
[in]	*OptionCount*	Number of option/value string pairs in OptionList.
[in]	*OptionList*	Pointer to array of option/value string pairs. Ignored if OptionCount is zero.
[out]	*PacketLength*	The number of bytes in the returned packet.
[out]	*Packet*	The pointer to the received packet. This buffer must be freed by the caller.

Return values:

EFI_SUCCESS	An MTFTPv6 OACK packet was received and is in the Packet. Note: It does not match the UEFI 2.3 Specification.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Filename is NULL. ◊ OptionCount is not zero and OptionList is NULL. ◊ One or more options in OptionList have wrong format. ◊ PacketLength is NULL. ◊ OverrideData.ServerIp is not a valid unicast IPv6 address.
EFI_UNSUPPORTED	One or more options in the OptionList are unsupported by this implementation.
EFI_NOT_STARTED	The EFI MTFTPv6 Protocol driver has not been started.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_ACCESS_DENIED	The previous operation has not completed yet.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_TFTP_ERROR	An MTFTPv6 ERROR packet was received and is in the Packet.
EFI_NETWORK_UNREACHABLE	An ICMP network unreachable error packet was received, and the Packet is set to NULL. Note: It is not defined in the UEFI 2.3 Specification.
EFI_HOST_UNREACHABLE	An ICMP host unreachable error packet was received, and the Packet is set to NULL. Note: It is not defined in the UEFI 2.3 Specification.
EFI_PROTOCOL_UNREACHABLE	An ICMP protocol unreachable error packet was received, and the Packet is set to NULL. Note: It is not defined in the UEFI 2.3 Specification.
EFI_PORT_UNREACHABLE	An ICMP port unreachable error packet was received, and the Packet is set to NULL.
EFI_ICMP_ERROR	Some other ICMP ERROR packet was received, and the Packet is set to NULL. Note: It does not match the UEFI 2.3 Specification.
EFI_PROTOCOL_ERROR	An unexpected MTFTPv6 packet was received and is in the Packet.
EFI_TIMEOUT	No responses were received from the MTFTPv6 server.
EFI_DEVICE_ERROR	An unexpected network error or system error occurred.

Get the information of the download from the server.

The GetInfo() function assembles an MTFTPv6 request packet with options, sends it to the MTFTPv6 server, and may return an MTFTPv6 OACK, MTFTPv6 ERROR, or ICMP ERROR packet. Retries occur only if no response packets are received from the MTFTPv6 server before the timeout expires.

Parameters:

[in]	*This*	Pointer to the EFI_MTFTP6_PROTOCOL instance.
[in]	*OverrideData*	Data that is used to override the existing parameters. If NULL, the default parameters that were set in the EFI_MTFTP6_PROTOCOL.Configure() function are used.
[in]	*Filename*	Pointer to null-terminated ASCII file name string.
[in]	*ModeStr*	Pointer to null-terminated ASCII mode string. If NULL, octet will be used.
[in]	*OptionCount*	Number of option/value string pairs in OptionList.
[in]	*OptionList*	Pointer to array of option/value string pairs. Ignored if OptionCount is zero.
[out]	*PacketLength*	The number of bytes in the returned packet.
[out]	*Packet*	The pointer to the received packet. This buffer must be freed by the caller.

Return values:

EFI_SUCCESS	An MTFTPv6 OACK packet was received and is in the Packet. Note: It does not match the UEFI 2.3 Specification.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Filename is NULL. ◊ OptionCount is not zero and OptionList is NULL. ◊ One or more options in OptionList have wrong format. ◊ PacketLength is NULL. ◊ OverrideData.ServerIp is not valid unicast IPv6 addresses.
EFI_UNSUPPORTED	One or more options in the OptionList are unsupported by this implementation.
EFI_NOT_STARTED	The EFI MTFTPv6 Protocol driver has not been started.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_ACCESS_DENIED	The previous operation has not completed yet.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_TFTP_ERROR	An MTFTPv6 ERROR packet was received and is in the Packet.
EFI_NETWORK_UNREACHABLE	An ICMP network unreachable error packet was received and the Packet is set to NULL. Note: It is not defined in UEFI 2.3 Specification.
EFI_HOST_UNREACHABLE	An ICMP host unreachable error packet was received and the Packet is set to NULL. Note: It is not defined in the UEFI 2.3 Specification.
EFI_PROTOCOL_UNREACHABLE	An ICMP protocol unreachable error packet was received and the Packet is set to NULL. Note: It is not defined in the UEFI 2.3 Specification.
EFI_PORT_UNREACHABLE	An ICMP port unreachable error packet was received and the Packet is set to NULL.
EFI_ICMP_ERROR	Some other ICMP ERROR packet was received and the Packet is set to NULL. Note: It does not match the UEFI 2.3 Specification.
EFI_PROTOCOL_ERROR	

	An unexpected MTFTPv6 packet was received and is in the Packet.
EFI_TIMEOUT	No responses were received from the MTFTPv6 server.
EFI_DEVICE_ERROR	An unexpected network error or system error occurred.
EFI_NO_MEDIA	There was a media error.

References Mtftp6CheckPacket(), Mtftp6OperationStart(), _MTFTP6_GETINFO_CONTEXT::Packet, _MTFTP6_GETINFO_CONTEXT::PacketLen, and _MTFTP6_GETINFO_CONTEXT::Status.

```
EFI_STATUS EFIAPI EfiMtftp6GetModeData ( IN EFI_MTFTP6_PROTOCOL *      This,
                                         OUT EFI_MTFTP6_MODE_DATA *    ModeData
                                       )
```

Returns the current operating mode data for the MTFTP6 instance.

The GetModeData() function returns the current operating mode and cached data packet for the MTFTP6 instance.

Parameters:

`[in]`	*This*	Pointer to the EFI_MTFTP6_PROTOCOL instance.
`[out]`	*ModeData*	The buffer in which the EFI MTFTPv6 Protocol driver mode data is returned.

Return values:

EFI_SUCCESS	The configuration data was returned successfully.
EFI_OUT_OF_RESOURCES	The required mode data could not be allocated.
EFI_INVALID_PARAMETER	This is NULL, or ModeData is NULL.

Returns the current operating mode data for the MTFTP6 instance.

The GetModeData() function returns the current operating mode and cached data packet for the MTFTP6 instance.

Parameters:

`[in]`	*This*	Pointer to the EFI_MTFTP6_PROTOCOL instance.
`[out]`	*ModeData*	The buffer in which the EFI MTFTPv6 Protocol driver mode data is returned.

Return values:

EFI_SUCCESS	The configuration data was returned successfully.
EFI_OUT_OF_RESOURCES	The required mode data could not be allocated.
EFI_INVALID_PARAMETER	This is NULL or ModeData is NULL.

References _MTFTP6_INSTANCE::Config, mMtftp6SupportedOptions, MTFTP6_INSTANCE_FROM_THIS, and MTFTP6_SUPPORTED_OPTIONS_NUM.

```
EFI_STATUS EFIAPI EfiMtftp6ParseOptions ( IN EFI_MTFTP6_PROTOCOL *   This,
                                          IN UINT32                  PacketLen,
                                          IN EFI_MTFTP6_PACKET *     Packet,
                                          OUT UINT32 *               OptionCount,
                                          OUT EFI_MTFTP6_OPTION **   OptionList
                                        )
```

Parse the options in an MTFTPv6 OACK packet.

The ParseOptions() function parses the option fields in an MTFTPv6 OACK packet and returns the number of options that were found, and optionally, a list of pointers to the options in the packet. If one or more of the option fields are not valid, then EFI_PROTOCOL_ERROR is returned and OptionCount and *OptionList stop at the last valid option.

Parameters:

`[in]`	*This*	Pointer to the EFI_MTFTP6_PROTOCOL instance.
`[in]`	*PacketLen*	Length of the OACK packet to be parsed.
`[in]`	*Packet*	Pointer to the OACK packet to be parsed.
`[out]`	*OptionCount*	Pointer to the number of options in the following OptionList.
`[out]`	*OptionList*	Pointer to EFI_MTFTP6_OPTION storage. Each pointer in the OptionList points to the corresponding MTFTP option buffer in the Packet. Call the EFI Boot Service FreePool() to release the OptionList if the options in this OptionList are not needed any more.

Return values:

EFI_SUCCESS	The OACK packet was valid and the OptionCount, and OptionList parameters have been updated.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ PacketLen is 0. ◊ Packet is NULL or Packet is not a valid MTFTPv6 packet. ◊ OptionCount is NULL.
EFI_NOT_FOUND	No options were found in the OACK packet.
EFI_OUT_OF_RESOURCES	Storage for the OptionList array can not be allocated.
EFI_PROTOCOL_ERROR	One or more of the option fields is invalid.

Parse the options in an MTFTPv6 OACK packet.

The ParseOptions() function parses the option fields in an MTFTPv6 OACK packet and returns the number of options that were found, and optionally, a list of pointers to the options in the packet. If one or more of the option fields are not valid, then EFI_PROTOCOL_ERROR is returned and OptionCount and *OptionList stop at the last valid option.

Parameters:

`[in]`	*This*	Pointer to the EFI_MTFTP6_PROTOCOL instance.
`[in]`	*PacketLen*	Length of the OACK packet to be parsed.
`[in]`	*Packet*	Pointer to the OACK packet to be parsed.
`[out]`	*OptionCount*	Pointer to the number of options in the following OptionList.
`[out]`	*OptionList*	Pointer to EFI_MTFTP6_OPTION storage. Each pointer in the OptionList points to the corresponding MTFTP option buffer in the Packet. Call the EFI Boot Service FreePool() to release the OptionList if the options in this OptionList are not needed anymore.

Return values:

EFI_SUCCESS	The OACK packet was valid and the OptionCount and OptionList parameters have been updated.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ PacketLen is 0. ◊ Packet is NULL or Packet is not a valid MTFTPv6 packet. ◊ OptionCount is NULL.
EFI_NOT_FOUND	No options were found in the OACK packet.
EFI_OUT_OF_RESOURCES	Storage for the OptionList array can not be allocated.
EFI_PROTOCOL_ERROR	One or more of the option fields is invalid.

References Mtftp6ParseStart().

EFI_STATUS EFIAPI EfiMtftp6Poll (IN EFI_MTFTP6_PROTOCOL * *This*)

Polls for incoming data packets and processes outgoing data packets.

The Poll() function can be used by network drivers and applications to increase the rate that data packets are moved between the communications device and the transmit and receive queues.In some systems, the periodic timer event in the managed network driver may not poll the underlying communications device fast enough to transmit and/or receive all data packets without missing incoming packets or dropping outgoing packets. Drivers and applications that are experiencing packet loss should try calling the Poll() function more often.

Parameters:

`[in]` *This* The MTFTP6 protocol instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_NOT_STARTED	This EFI MTFTPv6 Protocol instance has not been started.
EFI_INVALID_PARAMETER	This is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_TIMEOUT	Data was dropped out of the transmit and/or receive queue. Consider increasing the polling rate.

Polls for incoming data packets and processes outgoing data packets.

The Poll() function can be used by network drivers and applications to increase the rate that data packets are moved between the communications device and the transmit and receive queues. In some systems, the periodic timer event in the managed network driver may not poll the underlying communications device fast enough to transmit and/or receive all data packets without missing incoming packets or dropping outgoing packets. Drivers and applications that are experiencing packet loss should try calling the Poll() function more often.

Parameters:

`[in]` *This* The MTFTP6 protocol instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_NOT_STARTED	This EFI MTFTPv6 Protocol instance has not been started.
EFI_INVALID_PARAMETER	This is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_TIMEOUT	Data was dropped out of the transmit and/or receive queue. Consider increasing the polling rate.

References _MTFTP6_INSTANCE::Config, MTFTP6_INSTANCE_FROM_THIS, and _MTFTP6_INSTANCE::UdpIo.

EFI_STATUS EFIAPI EfiMtftp6ReadDirectory (IN EFI_MTFTP6_PROTOCOL * *This*,
IN EFI_MTFTP6_TOKEN * *Token*
)

Download a data file directory from an MTFTPv6 server.

The ReadDirectory() function is used to return a list of files on the MTFTPv6 server that are logically (or operationally) related to Token.Filename. The directory request packet that is sent to the server is built with the option list that was provided by caller, if present. The file information that the server returns is put into either of the following locations:

- A fixed buffer that is pointed to by Token.Buffer.
- A download service function that is pointed to by Token.CheckPacket. If both Token.Buffer and Token.CheckPacket are used, then Token.CheckPacket will be called first. If the call is successful, the packet will be stored in Token.Buffer.

Parameters:

`[in]` *This* Pointer to the EFI_MTFTP6_PROTOCOL instance.

`[in]` *Token* Pointer to the token structure to provide the parameters that are used in this operation.

Return values:

EFI_SUCCESS	The MTFTPv6 related file "directory" has been downloaded.
EFI_UNSUPPORTED	The EFI MTFTPv6 Protocol driver does not support this function.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token.Filename is NULL. ◊ Token.OptionCount is not zero and Token.OptionList is NULL. ◊ One or more options in Token.OptionList have wrong format. ◊ Token.Buffer and Token.CheckPacket are both NULL. ◊ Token.OverrideData.ServerIp is not valid unicast IPv6 addresses.
EFI_UNSUPPORTED	One or more options in the Token.OptionList are not supported by this implementation.
EFI_NOT_STARTED	The EFI MTFTPv6 Protocol driver has not been started.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_ALREADY_STARTED	This Token is already being used in another MTFTPv6 session.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_ACCESS_DENIED	The previous operation has not completed yet.
EFI_DEVICE_ERROR	An unexpected network error or system error occurred.

Download a data file directory from an MTFTPv6 server.

The ReadDirectory() function is used to return a list of files on the MTFTPv6 server that are logically (or operationally) related to Token.Filename. The directory request packet that is sent to the server is built with the option list that was provided by the caller, if present. The file information that the server returns is put into either of the following locations:

- A fixed buffer that is pointed to by Token.Buffer.
- A download service function that is pointed to by Token.CheckPacket. If both Token.Buffer and Token.CheckPacket are used, then Token.CheckPacket will be called first. If the call is successful, the packet will be stored in Token.Buffer.

Parameters:

`[in]` *This* Pointer to the EFI_MTFTP6_PROTOCOL instance.

`[in]` *Token* Pointer to the token structure to provide the parameters that are used in this operation.

Return values:

EFI_SUCCESS	The MTFTPv6 related file "directory" has been downloaded.
EFI_UNSUPPORTED	The EFI MTFTPv6 Protocol driver does not support this function.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token.Filename is NULL. ◊ Token.OptionCount is not zero and Token.OptionList is NULL. ◊ One or more options in Token.OptionList have wrong format. ◊ Token.Buffer and Token.CheckPacket are both NULL. ◊ Token.OverrideData.ServerIp is not valid unicast IPv6 addresses.
EFI_UNSUPPORTED	One or more options in the Token.OptionList are not supported by this implementation.
EFI_NOT_STARTED	The EFI MTFTPv6 Protocol driver has not been started.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_ALREADY_STARTED	This Token is already being used in another MTFTPv6 session.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_ACCESS_DENIED	The previous operation has not completed yet.
EFI_DEVICE_ERROR	An unexpected network error or system error occurred.

References MtftpOperationStart().

```
EFI_STATUS EFIAPI EfiMtftp6ReadFile ( IN EFI_MTFTP6_PROTOCOL *  This,
                                      IN EFI_MTFTP6_TOKEN *     Token
                                    )
```

Download a file from an MTFTPv6 server.

The ReadFile() function is used to initialize and start an MTFTPv6 download process and optionally wait for completion. When the download operation completes, whether successfully or not, the Token.Status field is updated by the EFI MTFTPv6 Protocol driver, and then Token.Event is signaled if it is not NULL. Data can be downloaded from the MTFTPv6 server into either of the following locations:

- A fixed buffer that is pointed to by Token.Buffer.
- A download service function that is pointed to by Token.CheckPacket. If both Token.Buffer and Token.CheckPacket are used, then Token.CheckPacket will be called first. If the call is successful, the packet will be stored in Token.Buffer.

Parameters:

[in]	*This*	Pointer to the EFI_MTFTP6_PROTOCOL instance.
[in]	*Token*	Pointer to the token structure to provide the parameters that are used in this operation.

Return values:

EFI_SUCCESS	The data file has been transferred successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_BUFFER_TOO_SMALL	BufferSize is not zero but not large enough to hold the downloaded data in downloading process. Note: It does not match the UEFI 2.3 Specification.
EFI_ABORTED	Current operation is aborted by user.

EFI_NETWORK_UNREACHABLE	An ICMP network unreachable error packet was received. Note: It is not defined in the UEFI 2.3 Specification.
EFI_HOST_UNREACHABLE	An ICMP host unreachable error packet was received. Note: It is not defined in the UEFI 2.3 Specification.
EFI_PROTOCOL_UNREACHABLE	An ICMP protocol unreachable error packet was received. Note: It is not defined in the UEFI 2.3 Specification.
EFI_PORT_UNREACHABLE	An ICMP port unreachable error packet was received. Note: It is not defined in the UEFI 2.3 Specification.
EFI_ICMP_ERROR	An ICMP ERROR packet was received.
EFI_TIMEOUT	No responses were received from the MTFTPv6 server.
EFI_TFTP_ERROR	An MTFTPv6 ERROR packet was received.
EFI_DEVICE_ERROR	An unexpected network error or system error occurred.

Download a file from an MTFTPv6 server.

The ReadFile() function is used to initialize and start an MTFTPv6 download process, and optionally, wait for completion. When the download operation completes, whether successfully or not, the Token.Status field is updated by the EFI MTFTPv6 Protocol driver, and then Token.Event is signaled if it is not NULL. Data can be downloaded from the MTFTPv6 server into either of the following locations:

- A fixed buffer that is pointed to by Token.Buffer
- A download service function that is pointed to by Token.CheckPacket. If both Token.Buffer and Token.CheckPacket are used, then Token.CheckPacket will be called first. If the call is successful, the packet will be stored in Token.Buffer.

Parameters:

`[in]`	*This*	Pointer to the EFI_MTFTP6_PROTOCOL instance.
`[in]`	*Token*	Pointer to the token structure to provide the parameters that are used in this operation.

Return values:

EFI_SUCCESS	The data file has been transferred successfully.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_BUFFER_TOO_SMALL	BufferSize is not zero but not large enough to hold the downloaded data in downloading process. Note: It does not match the UEFI 2.3 Specification.
EFI_ABORTED	Current operation is aborted by user.
EFI_NETWORK_UNREACHABLE	An ICMP network unreachable error packet was received. Note: It is not defined in the UEFI 2.3 Specification.
EFI_HOST_UNREACHABLE	An ICMP host unreachable error packet was received. Note: It is not defined in the UEFI 2.3 Specification.
EFI_PROTOCOL_UNREACHABLE	An ICMP protocol unreachable error packet was received. Note: It is not defined in the UEFI 2.3 Specification.
EFI_PORT_UNREACHABLE	An ICMP port unreachable error packet was received. Note: It is not defined in the UEFI 2.3 Specification.
EFI_ICMP_ERROR	An ICMP ERROR packet was received.
EFI_TIMEOUT	No responses were received from the MTFTPv6 server.
EFI_TFTP_ERROR	An MTFTPv6 ERROR packet was received.
EFI_DEVICE_ERROR	An unexpected network error or system error occurred.
EFI_NO_MEDIA	There was a media error.

References Mtftp6OperationStart().

```
EFI_STATUS EFIAPI EfiMtftp6WriteFile ( IN EFI_MTFTP6_PROTOCOL *  This,
                                       IN EFI_MTFTP6_TOKEN *     Token
                                     )
```

Send a file to an MTFTPv6 server.

The WriteFile() function is used to initialize an uploading operation with the given option list, and optionally, wait for completion. If one or more of the options is not supported by the server, the unsupported options are ignored and a standard TFTP process starts instead. When the upload process completes, whether successfully or not, Token.Event is signaled, and the EFI MTFTPv6 Protocol driver updates Token.Status. The caller can supply the data to be uploaded in the following two modes:

- Through the user-provided buffer.
- Through a callback function. With the user-provided buffer, the Token.BufferSize field indicates the length of the buffer, and the driver will upload the data in the buffer. With an EFI_MTFTP6_PACKET_NEEDED callback function, the driver will call this callback function to get more data from the user to upload.

Parameters:

`[in]` *This* Pointer to the EFI_MTFTP6_PROTOCOL instance.

`[in]` *Token* Pointer to the token structure to provide the parameters that are used in this operation.

Return values:

EFI_SUCCESS	The upload session has started.
EFI_UNSUPPORTED	The operation is not supported by this implementation.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token.Filename is NULL. ◊ Token.OptionCount is not zero and Token.OptionList is NULL. ◊ One or more options in Token.OptionList have wrong format. ◊ Token.Buffer and Token.PacketNeeded are both NULL. ◊ Token.OverrideData.ServerIp is not valid unicast IPv6 addresses.
EFI_UNSUPPORTED	One or more options in the Token.OptionList are not supported by this implementation.
EFI_NOT_STARTED	The EFI MTFTPv6 Protocol driver has not been started.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_ALREADY_STARTED	This Token is already being used in another MTFTPv6 session.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_ACCESS_DENIED	The previous operation has not completed yet.
EFI_DEVICE_ERROR	An unexpected network error or system error occurred.

Send a file to an MTFTPv6 server.

The WriteFile() function is used to initialize an uploading operation with the given option list and optionally wait for completion. If one or more of the options is not supported by the server, the unsupported options are ignored and a standard TFTP process starts instead. When the upload process completes, whether successfully or not, Token.Event is signaled, and the EFI MTFTPv6 Protocol driver updates Token.Status. The caller can supply the data to be uploaded in the following two modes:

- Through the user-provided buffer

- Through a callback function With the user-provided buffer, the Token.BufferSize field indicates the length of the buffer, and the driver will upload the data in the buffer. With an EFI_MTFTP6_PACKET_NEEDED callback function, the driver will call this callback function to get more data from the user to upload.

Parameters:

`[in]` *This* Pointer to the EFI_MTFTP6_PROTOCOL instance.

`[in]` *Token* Pointer to the token structure to provide the parameters that are used in this operation.

Return values:

EFI_SUCCESS	The upload session has started.
EFI_UNSUPPORTED	The operation is not supported by this implementation.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token.Filename is NULL. ◊ Token.OptionCount is not zero and Token.OptionList is NULL. ◊ One or more options in Token.OptionList have wrong format. ◊ Token.Buffer and Token.PacketNeeded are both NULL. ◊ Token.OverrideData.ServerIp is not a valid unicast IPv6 address.
EFI_UNSUPPORTED	One or more options in the Token.OptionList are not supported by this implementation.
EFI_NOT_STARTED	The EFI MTFTPv6 Protocol driver has not been started.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_ALREADY_STARTED	This Token is already being used in another MTFTPv6 session.
EFI_OUT_OF_RESOURCES	Required system resources could not be allocated.
EFI_ACCESS_DENIED	The previous operation has not completed yet.
EFI_DEVICE_ERROR	An unexpected network error or system error occurred.

References Mtftp6OperationStart().

Variable Documentation

EFI_MTFTP6_PROTOCOL gMtftp6ProtocolTemplate

Referenced by Mtftp6CreateInstance().

NetworkPkg/Mtftp6Dxe/Mtftp6Option.c File Reference

Functions

EFI_STATUS Mtftp6ParseMcastOption (IN UINT8 *Str, IN MTFTP6_EXT_OPTION_INFO *ExtInfo)
EFI_STATUS Mtftp6ParseExtensionOption (IN EFI_MTFTP6_OPTION *Options, IN UINT32 Count, IN BOOLEAN IsRequest, IN MTFTP6_EXT_OPTION_INFO *ExtInfo)
EFI_STATUS Mtftp6ParsePacketOption (IN EFI_MTFTP6_PACKET *Packet, IN UINT32 PacketLen, IN OUT UINT32 *Count, IN EFI_MTFTP6_OPTION *Options)
EFI_STATUS Mtftp6ParseStart (IN EFI_MTFTP6_PACKET *Packet, IN UINT32 PacketLen, IN OUT UINT32 *OptionCount, OUT EFI_MTFTP6_OPTION **OptionList)

Variables

CHAR8 * mMtftp6SupportedOptions [MTFTP6_SUPPORTED_OPTIONS_NUM]

Detailed Description

Mtftp6 option parse functions implementation.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS Mtftp6ParseExtensionOption ( IN EFI_MTFTP6_OPTION *          Options,
                                        IN UINT32                       Count,
                                        IN BOOLEAN                      IsRequest,
                                        IN MTFTP6_EXT_OPTION_INFO *     ExtInfo
                                      )
```

Parse the MTFTP6 extesion options.

Parameters:

[in]	*Options*	The pointer to the extension options list.
[in]	*Count*	The num of the extension options.
[in]	*IsRequest*	If FALSE, the extension options is included by a request packet.
[in]	*ExtInfo*	The pointer to the option information to be filled.

Return values:

EFI_SUCCESS	Parse the multicast option successfully.
EFI_INVALID_PARAMETER	There is one option is malformatted at least.
EFI_UNSUPPORTED	There is one option is not supported at least.

References MTFTP6_OPT_BLKSIZE_BIT, MTFTP6_OPT_MCAST_BIT, MTFTP6_OPT_TIMEOUT_BIT, MTFTP6_OPT_TSIZE_BIT, and Mtftp6ParseMcastOption().

Referenced by Mtftp6OperationStart(), Mtftp6RrqHandleOack(), and Mtftp6WrqHandleOack().

```
EFI_STATUS Mtftp6ParseMcastOption ( IN UINT8 *                    Str,
                                    IN MTFTP6_EXT_OPTION_INFO *   ExtInfo
                                  )
```

Parse the NULL terminated ASCII string of multicast option.

Parameters:

[in]	*Str*	The pointer to the Ascii string of multicast option.
[in]	*ExtInfo*	The pointer to the option information to be filled.

Return values:

EFI_SUCCESS	Parse the multicast option successfully.
EFI_INVALID_PARAMETER	The string is malformatted.
EFI_OUT_OF_RESOURCES	Failed to perform the operation due to lack of resources.

Referenced by Mtftp6ParseExtensionOption().

```
EFI_STATUS Mtftp6ParsePacketOption ( IN EFI_MTFTP6_PACKET *   Packet,
                                     IN UINT32                PacketLen,
                                     IN OUT UINT32 *          Count,
                                     IN EFI_MTFTP6_OPTION *   Options
                                   )
```

Go through the packet to fill the options array with the start addresses of each MTFTP option name/value pair.

Parameters:

[in]	*Packet*	The packet to be checked.
[in]	*PacketLen*	The length of the packet.
[in,out]	*Count*	The num of the Options on input. The actual one on output.
[in]	*Options*	The option array to be filled. It is optional.

Return values:

EFI_SUCCESS	The packet has been parsed successfully.
EFI_INVALID_PARAMETER	The packet is malformatted.
EFI_BUFFER_TOO_SMALL	The Options array is too small.
EFI_PROTOCOL_ERROR	An unexpected MTFTPv6 packet was received.

References MTFTP6_OPCODE_LEN.

Referenced by Mtftp6ParseStart().

```
EFI_STATUS Mtftp6ParseStart ( IN EFI_MTFTP6_PACKET *        Packet,
                              IN UINT32                     PacketLen,
                              IN OUT UINT32 *               OptionCount,
                              OUT EFI_MTFTP6_OPTION **      OptionList
                            )
```

Go through the packet, generate option list array and fill it by the result of parse options.

Parameters:

[in]	*Packet*	The packet to be checked.
[in]	*PacketLen*	The length of the packet.
[in,out]	*OptionCount*	The num of the Options on input. The actual one on output.
[out]	*OptionList*	The option list array to be generated and filled. It is optional.

Return values:

EFI_SUCCESS	The packet has been parsed successfully.
EFI_INVALID_PARAMETER	The packet is malformatted.
EFI_PROTOCOL_ERROR	There is one option is malformatted at least.
EFI_NOT_FOUND	The packet has no options.
EFI_OUT_OF_RESOURCES	Failed to allocate memory for the array.
EFI_BUFFER_TOO_SMALL	The size of option list array is too small.

References MtftpPacketOption().

Referenced by EfiMtftp6ParseOptions(), Mtftp6RrqHandleOack(), and Mtftp6WrqHandleOack().

Variable Documentation

CHAR8* mMtftp6SupportedOptions[MTFTP6_SUPPORTED_OPTIONS_NUM]

Initial value:

```
 {
  "blksize",
  "timeout",
  "tsize",
  "multicast"
}
```

Referenced by EfiMtftp6GetModeData().

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Mtftp6Dxe/Mtftp6Option.h File Reference

Data Structures

struct MTFTP6_EXT_OPTION_INFO

Defines

"memItemRight" valign="bottom">MTFTP6_SUPPORTED_OPTIONS_NUM 4

#define MTFTP6_OPCODE_LEN 2
#define MTFTP6_ERRCODE_LEN 2
#define MTFTP6_BLKNO_LEN 2
#define MTFTP6_DATA_HEAD_LEN 4
#define MTFTP6_OPT_BLKSIZE_BIT 0
#define MTFTP6_OPT_TIMEOUT_BIT
#define MTFTP6_OPT_TSIZE_BIT 0x04
#define MTFTP6_OPT_MCAST_BIT 0x

Functions

EFI_STATUS Mtftp6ParseMcastOption (IN UINT
MTFTP6_EXT_OPTION_INFO *
EFI_STATUS Mtftp6ParseExtensionOption (IN
EFI_MTFTP6_OPTION *Options,
UINT32 Count, IN BOOLEAN IsR
MTFTP6_EXT_OPTION_INFO *
EFI_STATUS Mtftp6ParsePacketOption (IN
EFI_MTFTP6_PACKET *Packet,
UINT32 PacketLen, IN OUT UINT
*Count, IN EFI_MTFTP6_OPTIO
*Options)
EFI_STATUS Mtftp6ParseStart (IN
EFI_MTFTP6_PACKET *Packet,
UINT32 PacketLen, IN OUT UINT
*OptionCount, OUT EFI_MTFTP6
**OptionList)

Variables

CHAR8 * mMtftp6SupportedOptions
[MTFTP6_SUPPORTED_OPTION

Detailed Description

Mtftp6 option parse functions declaration.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Mtftp6WrqInput().

#define MTFTP6_DATA_HEAD_LEN 4

Referenced by Mtftp6RrqInput(), Mtftp6RrqSaveBlock(), and Mtftp6WrqSendBlock().

#define MTFTP6_ERRCODE_LEN 2

#define MTFTP6_OPCODE_LEN 2

Referenced by Mtftp6ParsePacketOption(), Mtftp6RrqInput(), and Mtftp6WrqInput().

#define MTFTP6_OPT_BLKSIZE_BIT 0x01

Referenced by Mtftp6ParseExtensionOption(), Mtftp6RrqOackValid(), and Mtftp6WrqOackValid().

#define MTFTP6_OPT_MCAST_BIT 0x08

Referenced by Mtftp6ParseExtensionOption(), Mtftp6RrqHandleOack(), and Mtftp6RrqOackValid().

#define MTFTP6_OPT_TIMEOUT_BIT 0x02

Referenced by Mtftp6ParseExtensionOption(), Mtftp6RrqOackValid(), and Mtftp6WrqOackValid().

#define MTFTP6_OPT_TSIZE_BIT 0x04

Referenced by Mtftp6ParseExtensionOption().

#define MTFTP6_SUPPORTED_OPTIONS_NUM 4

Referenced by EfiMtftp6GetModeData().

Function Documentation

```
EFI_STATUS Mtftp6ParseExtensionOption ( IN EFI_MTFTP6_OPTION *          Options,
                                        IN UINT32                       Count,
                                        IN BOOLEAN                      IsRequest,
                                        IN MTFTP6_EXT_OPTION_INFO *     ExtInfo
                                      )
```

Parse the MTFTP6 extesion options.

Parameters:

[in]	*Options*	The pointer to the extension options list.
[in]	*Count*	The num of the extension options.
[in]	*IsRequest*	If FALSE, the extension options is included by a request packet.
[in]	*ExtInfo*	The pointer to the option information to be filled.

Return values:

EFI_SUCCESS	Parse the multi-cast option successfully.
EFI_INVALID_PARAMETER	An option is malformatted.
EFI_UNSUPPORTED	An option is not supported.

Parse the MTFTP6 extesion options.

Parameters:

[in]	*Options*	The pointer to the extension options list.
[in]	*Count*	The num of the extension options.
[in]	*IsRequest*	If FALSE, the extension options is included by a request packet.
[in]	*ExtInfo*	The pointer to the option information to be filled.

Return values:

EFI_SUCCESS	Parse the multicast option successfully.
EFI_INVALID_PARAMETER	There is one option is malformatted at least.
EFI_UNSUPPORTED	There is one option is not supported at least.

References MTFTP6_OPT_BLKSIZE_BIT, MTFTP6_OPT_MCAST_BIT, MTFTP6_OPT_TIMEOUT_BIT, MTFTP6_OPT_TSIZE_BIT, and Mtftp6ParseMcastOption().

Referenced by Mtftp6OperationStart(), Mtftp6RrqHandleOack(), and Mtftp6WrqHandleOack().

```
EFI_STATUS Mtftp6ParseMcastOption ( IN UINT8 *                  Str,
                                    IN MTFTP6_EXT_OPTION_INFO * ExtInfo
                                  )
```

Parse the Ascii string of multi-cast option.

Parameters:

[in]	*Str*	The pointer to the Ascii string of multi-cast option.
[in]	*ExtInfo*	The pointer to the option information to be filled.

Return values:

EFI_SUCCESS	Parse the multicast option successfully.

EFI_INVALID_PARAMETER The string is malformatted.

Parse the NULL terminated ASCII string of multicast option.

Parameters:

`[in]` *Str* The pointer to the Ascii string of multicast option.
`[in]` *ExtInfo* The pointer to the option information to be filled.

Return values:

EFI_SUCCESS Parse the multicast option successfully.
EFI_INVALID_PARAMETER The string is malformatted.
EFI_OUT_OF_RESOURCES Failed to perform the operation due to lack of resources.

Referenced by Mtftp6ParseExtensionOption().

```
EFI_STATUS Mtftp6ParsePacketOption ( IN EFI_MTFTP6_PACKET *   Packet,
                                     IN UINT32                PacketLen,
                                     IN OUT UINT32 *          Count,
                                     IN EFI_MTFTP6_OPTION *   Options
                                   )
```

Go through the packet to fill the options array with the start addresses of each MTFTP option name/value pair.

Parameters:

`[in]` *Packet* The packet to be checked.
`[in]` *PacketLen* The length of the packet.
`[in,out]` *Count* The num of the Options on input. The actual one on output.
`[in]` *Options* The option array to be filled it's optional.

Return values:

EFI_SUCCESS The packet has been parsed successfully.
EFI_INVALID_PARAMETER The packet is malformatted
EFI_BUFFER_TOO_SMALL The Options array is too small
EFI_PROTOCOL_ERROR An unexpected MTFTPv6 packet was received.

Go through the packet to fill the options array with the start addresses of each MTFTP option name/value pair.

Parameters:

`[in]` *Packet* The packet to be checked.
`[in]` *PacketLen* The length of the packet.
`[in,out]` *Count* The num of the Options on input. The actual one on output.
`[in]` *Options* The option array to be filled. It is optional.

Return values:

EFI_SUCCESS The packet has been parsed successfully.
EFI_INVALID_PARAMETER The packet is malformatted.
EFI_BUFFER_TOO_SMALL The Options array is too small.
EFI_PROTOCOL_ERROR An unexpected MTFTPv6 packet was received.

References MTFTP6_OPCODE_LEN.

Referenced by Mtftp6ParseStart().

```
EFI_STATUS Mtftp6ParseStart ( IN EFI_MTFTP6_PACKET *        Packet,
                              IN UINT32                     PacketLen,
                              IN OUT UINT32 *               OptionCount,
                              OUT EFI_MTFTP6_OPTION **      OptionList
                            )
```

Go through the packet, generate option list array and fill it by the result of parse options.

Parameters:

[in]	*Packet*	The packet to be checked.
[in]	*PacketLen*	The length of the packet.
[in,out]	*OptionCount*	The num of the Options on input. The actual one on output.
[out]	*OptionList*	The option list array to be generated and filled. It is optional.

Return values:

EFI_SUCCESS	The packet has been parsed successfully.
EFI_INVALID_PARAMETER	The packet is malformatted.
EFI_PROTOCOL_ERROR	An option is malformatted.
EFI_NOT_FOUND	The packet has no options.
EFI_OUT_OF_RESOURCES	Failed to allocate memory for the array.
EFI_BUFFER_TOO_SMALL	The size of option list array is too small.

Go through the packet, generate option list array and fill it by the result of parse options.

Parameters:

[in]	*Packet*	The packet to be checked.
[in]	*PacketLen*	The length of the packet.
[in,out]	*OptionCount*	The num of the Options on input. The actual one on output.
[out]	*OptionList*	The option list array to be generated and filled. It is optional.

Return values:

EFI_SUCCESS	The packet has been parsed successfully.
EFI_INVALID_PARAMETER	The packet is malformatted.
EFI_PROTOCOL_ERROR	There is one option is malformatted at least.
EFI_NOT_FOUND	The packet has no options.
EFI_OUT_OF_RESOURCES	Failed to allocate memory for the array.
EFI_BUFFER_TOO_SMALL	The size of option list array is too small.

References MtftpPacketOption().

Referenced by EfiMtftpParseOptions(), MtftpRrqHandleOack(), and MtftpWrqHandleOack().

Variable Documentation

CHAR8* mMtftpSupportedOptions[MTFTP6_SUPPORTED_OPTIONS_NUM]

Referenced by EfiMtftpGetModeData().

- Data Structures
- Files

- File List
- Globals

NetworkPkg/Mtftp6Dxe/Mtftp6Rrq.c File Reference

Functions

EFI_STATUS Mtftp6RrqSendAck (IN MTFTP6_INSTANCE *Instance, IN UINT16 BlockNum)
EFI_STATUS Mtftp6RrqSaveBlock (IN MTFTP6_INSTANCE *Instance, IN EFI_MTFTP6_PACKET *Packet, IN UINT32 Len, OUT NET_BUF **UdpPacket)
EFI_STATUS Mtftp6RrqHandleData (IN MTFTP6_INSTANCE *Instance, IN EFI_MTFTP6_PACKET *Packet, IN UINT32 Len, OUT NET_BUF **UdpPacket, OUT BOOLEAN *IsCompleted)
BOOLEAN Mtftp6RrqOackValid (IN MTFTP6_INSTANCE *Instance, IN MTFTP6_EXT_OPTION_INFO *ReplyInfo, IN MTFTP6_EXT_OPTION_INFO *RequestInfo)
EFI_STATUS EFIAPI Mtftp6RrqConfigMcastUdpIo (IN UDP_IO *McastIo, IN VOID *Context)
EFI_STATUS Mtftp6RrqHandleOack (IN MTFTP6_INSTANCE *Instance, IN EFI_MTFTP6_PACKET *Packet, IN UINT32 Len, OUT NET_BUF **UdpPacket, OUT BOOLEAN *IsCompleted)
VOID EFIAPI Mtftp6RrqInput (IN NET_BUF *UdpPacket, IN UDP_END_POINT *UdpEpt, IN EFI_STATUS IoStatus, IN VOID *Context)
EFI_STATUS Mtftp6RrqStart (IN MTFTP6_INSTANCE *Instance, IN UINT16 Operation)

Detailed Description

Mtftp6 Rrq process functions implementation.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI Mtftp6RrqConfigMcastUdpIo (IN UDP_IO * *McastIo*, IN VOID * *Context*)

Configure Udp6Io to receive a packet from a multicast address.

Parameters:

[in] *McastIo* The pointer to the mcast Udp6Io.
[in] *Context* The pointer to the context.

Return values:

EFI_SUCCESS The mcast Udp6Io was successfully configured.
Others Failed to configure the Udp6Io.

References _MTFTP6_INSTANCE::McastIp, _MTFTP6_INSTANCE::McastPort, and _MTFTP6_INSTANCE::ServerIp.

Referenced by Mtftp6RrqHandleOack().

```
EFI_STATUS Mtftp6RrqHandleData ( IN MTFTP6_INSTANCE *          Instance,
                                 IN EFI_MTFTP6_PACKET *        Packet,
                                 IN UINT32                     Len,
                                 OUT NET_BUF **                UdpPacket,
                                 OUT BOOLEAN *                 IsCompleted
                               )
```

Process the received data packets. It will save the block then send back an ACK if it is active.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*Packet*	The pointer to the received packet.
[in]	*Len*	The length of the packet.
[out]	*UdpPacket*	The net buf of received packet.
[out]	*IsCompleted*	If TRUE, the download has been completed. Otherwise, the download has not been completed.

Return values:

EFI_SUCCESS	The data packet was successfully processed.
EFI_ABORTED	The download was aborted by the user.
EFI_BUFFER_TOO_SMALL	The user-provided buffer is too small.

References Mtftp6GetNextBlockNum(), Mtftp6RrqSaveBlock(), Mtftp6RrqSendAck(), and Mtftp6TransmitPacket().

Referenced by Mtftp6RrqInput().

```
EFI_STATUS Mtftp6RrqHandleOack ( IN MTFTP6_INSTANCE *          Instance,
                                 IN EFI_MTFTP6_PACKET *        Packet,
                                 IN UINT32                     Len,
                                 OUT NET_BUF **                UdpPacket,
                                 OUT BOOLEAN *                 IsCompleted
                               )
```

Process the OACK packet for Rrq.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*Packet*	The pointer to the received packet.
[in]	*Len*	The length of the packet.
[out]	*UdpPacket*	The net buf of received packet.
[out]	*IsCompleted*	If TRUE, the download has been completed. Otherwise, the download has not been completed.

Return values:

EFI_DEVICE_ERROR	Failed to create/start a multicast Udp6 child.
EFI_TFTP_ERROR	An error happened during the process.
EFI_SUCCESS	The OACK packet successfully processed.

References MTFTP6_EXT_OPTION_INFO::BitMap, MTFTP6_EXT_OPTION_INFO::BlkSize, MTFTP6_EXT_OPTION_INFO::IsMaster, MTFTP6_EXT_OPTION_INFO::McastIp, MTFTP6_EXT_OPTION_INFO::McastPort, MTFTP6_OPT_MCAST_BIT, Mtftp6GetNextBlockNum(), Mtftp6ParseExtensionOption(), Mtftp6ParseStart(), Mtftp6RrqConfigMcastUdpIo(), Mtftp6RrqInput(), Mtftp6RrqOackValid(), Mtftp6RrqSendAck(), Mtftp6SendError(), and MTFTP6_EXT_OPTION_INFO::Timeout.

Referenced by Mtftp6RrqInput().

```
VOID EFIAPI Mtftp6RrqInput ( IN NET_BUF *          UdpPacket,
                             IN UDP_END_POINT *    UdpEpt,
                             IN EFI_STATUS         IoStatus,
                             IN VOID *             Context
                           )
```

The packet process callback for Mtftp6 download.

Parameters:

[in]	*UdpPacket*	The pointer to the packet received.
[in]	*UdpEpt*	The pointer to the Udp6 access point.
[in]	*IoStatus*	The status from Udp6 instance.
[in]	*Context*	The pointer to the context.

References _MTFTP6_INSTANCE::BlkSize, _MTFTP6_INSTANCE::McastIp, _MTFTP6_INSTANCE::McastUdpIo, _MTFTP6_INSTANCE::Mtftp6, MTFTP6_DATA_HEAD_LEN, MTFTP6_INSTANCE_SIGNATURE, MTFTP6_OPCODE_LEN, Mtftp6OperationClean(), Mtftp6RrqHandleData(), Mtftp6RrqHandleOack(), Mtftp6RrqInput(), Mtftp6SendError(), _MTFTP6_INSTANCE::ServerDataPort, _MTFTP6_INSTANCE::Token, and _MTFTP6_INSTANCE::UdpIo.

Referenced by Mtftp6RrqHandleOack(), Mtftp6RrqInput(), and Mtftp6RrqStart().

```
BOOLEAN Mtftp6RrqOackValid ( IN MTFTP6_INSTANCE *          Instance,
                             IN MTFTP6_EXT_OPTION_INFO *   ReplyInfo,
                             IN MTFTP6_EXT_OPTION_INFO *   RequestInfo
                           )
```

Validate whether the options received in the server's OACK packet is valid. The options are valid only if: 1. The server doesn't include options not requested by us. 2. The server can only use smaller blksize than that is requested. 3. The server can only use the same timeout as requested. 4. The server doesn't change its multicast channel.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*ReplyInfo*	The pointer to options information in reply packet.
[in]	*RequestInfo*	The pointer to requested options info.

Return values:

TRUE	If the option in the OACK is valid.
FALSE	If the option is invalid.

References MTFTP6_OPT_BLKSIZE_BIT, MTFTP6_OPT_MCAST_BIT, and MTFTP6_OPT_TIMEOUT_BIT.

Referenced by Mtftp6RrqHandleOack().

```
EFI_STATUS Mtftp6RrqSaveBlock ( IN MTFTP6_INSTANCE *      Instance,
                                IN EFI_MTFTP6_PACKET *    Packet,
                                IN UINT32                 Len,
                                OUT NET_BUF **            UdpPacket
                              )
```

Deliver the received data block to the user, which can be saved in the user provide buffer or through the CheckPacket callback.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*Packet*	The pointer to the received packet.
[in]	*Len*	The packet length.
[out]	*UdpPacket*	The net buf of the received packet.

Return values:

EFI_SUCCESS	The data was saved successfully.
EFI_ABORTED	The user tells to abort by return an error through CheckPacket.
EFI_BUFFER_TOO_SMALL	The user's buffer is too small, and buffer length is updated to the actual buffer size needed.

References MTFTP6_DATA_HEAD_LEN, Mtftp6RemoveBlockNum(), Mtftp6SendError(), and Mtftp6SetLastBlockNum().

Referenced by Mtftp6RrqHandleData().

```
EFI_STATUS Mtftp6RrqSendAck ( IN MTFTP6_INSTANCE *   Instance,
                              IN UINT16              BlockNum
                            )
```

Build and send a ACK packet for download.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*BlockNum*	The block number to be acked.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate memory for the packet.
EFI_SUCCESS	The ACK has been sent.
Others	Failed to send the ACK.

References Mtftp6TransmitPacket().

Referenced by Mtftp6RrqHandleData(), and Mtftp6RrqHandleOack().

```
EFI_STATUS Mtftp6RrqStart ( IN MTFTP6_INSTANCE *   Instance,
                            IN UINT16              Operation
                          )
```

Start the Mtftp6 instance to download. It first initializes some of the internal states, then builds and sends an RRQ reqeuest packet. Finally, it starts receive for the downloading.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*Operation*	The operation code of current packet.

Return values:

EFI_SUCCESS	The Mtftp6 is started to download.
Others	Failed to start to download.

References Mtftp6InitBlockRange(), Mtftp6RrqInput(), and Mtftp6SendRequest().

Referenced by Mtftp6OperationStart().

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Mtftp6Dxe/Mtftp6Support.c File Reference

Functions

MTFTP6_BLOCK_RANGE * Mtftp6AllocateRange (IN UINT16 Start, IN UINT16 End)
EFI_STATUS Mtftp6InitBlockRange (IN LIST_ENTRY *Head, IN UINT16 Start, IN UINT16 End)
INTN Mtftp6GetNextBlockNum (IN LIST_ENTRY *Head)
VOID Mtftp6SetLastBlockNum (IN LIST_ENTRY *Head, IN UINT16 Last)
EFI_STATUS Mtftp6RemoveBlockNum (IN LIST_ENTRY *Head, IN UINT16 Num, IN BOOLEAN Completed, OUT UINT64 *TotalBlock)
EFI_STATUS Mtftp6GetMapping (IN UDP_IO *UdpIo, IN EFI_UDP6_CONFIG_DATA *UdpCfgData)
EFI_STATUS EFIAPI Mtftp6ConfigDummyUdpIo (IN UDP_IO *UdpIo, IN VOID *Context)
EFI_STATUS Mtftp6ConfigUdpIo (IN UDP_IO *UdpIo, IN EFI_IPv6_ADDRESS *ServerIp, IN UINT16 ServerPort, IN EFI_IPv6_ADDRESS *LocalIp, IN UINT16 LocalPort)
EFI_STATUS Mtftp6SendRequest (IN MTFTP6_INSTANCE *Instance, IN UINT16 Operation)
EFI_STATUS Mtftp6SendError (IN MTFTP6_INSTANCE *Instance, IN UINT16 ErrCode, IN UINT8 *ErrInfo)
VOID EFIAPI Mtftp6OnPacketSent (IN NET_BUF *Packet, IN UDP_END_POINT *UdpEpt, IN EFI_STATUS IoStatus, IN VOID *Context)
EFI_STATUS Mtftp6TransmitPacket (IN MTFTP6_INSTANCE *Instance, IN NET_BUF *Packet)
EFI_STATUS EFIAPI Mtftp6CheckPacket (IN EFI_MTFTP6_PROTOCOL *This, IN EFI_MTFTP6_TOKEN *Token, IN UINT16 PacketLen, IN EFI_MTFTP6_PACKET *Packet)
VOID Mtftp6OperationClean (IN MTFTP6_INSTANCE *Instance, IN EFI_STATUS Result)
EFI_STATUS Mtftp6OperationStart (IN EFI_MTFTP6_PROTOCOL *This, IN EFI_MTFTP6_TOKEN *Token, IN UINT16 OpCode)
VOID EFIAPI Mtftp6OnTimerTick (IN EFI_EVENT Event, IN VOID *Context)

Detailed Description

Mtftp6 support functions implementation.

Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
MTFTP6_BLOCK_RANGE* Mtftp6AllocateRange ( IN UINT16 Start,
                                          IN UINT16 End
                                        )
```

Allocate a MTFTP block range, then init it to the range of [Start, End].

Parameters:

[in] *Start* The start block number.

[in] *End* The last block number in the range.

Returns:

Range The range of the allocated block buffer.

References MTFTP6_BLOCK_RANGE::Bound, MTFTP6_BLOCK_RANGE::End, MTFTP6_BLOCK_RANGE::Link, and MTFTP6_BLOCK_RANGE::Start.

Referenced by Mtftp6InitBlockRange(), and Mtftp6RemoveBlockNum().

```
EFI_STATUS EFIAPI Mtftp6CheckPacket ( IN EFI_MTFTP6_PROTOCOL * This,
                                      IN EFI_MTFTP6_TOKEN * Token,
                                      IN UINT16 PacketLen,
                                      IN EFI_MTFTP6_PACKET * Packet
                                    )
```

Check packet for GetInfo callback routine.

GetInfo is implemented with EfiMtftp6ReadFile. It's used to inspect the first packet from server, then abort the session.

Parameters:

[in] *This* The pointer to the Mtftp6 protocol.

[in] *Token* The pointer to the Mtftp6 token.

[in] *PacketLen* The length of the packet.

[in] *Packet* The pointer to the received packet.

Return values:

EFI_ABORTED Abort the Mtftp6 operation.

References _MTFTP6_GETINFO_CONTEXT::Packet, _MTFTP6_GETINFO_CONTEXT::PacketLen, and _MTFTP6_GETINFO_CONTEXT::Status.

Referenced by EfiMtftp6GetInfo().

```
EFI_STATUS EFIAPI Mtftp6ConfigDummyUdpIo ( IN UDP_IO * UdpIo,
                                           IN VOID * Context
                                         )
```

The dummy configure routine for create a new Udp6 Io.

Parameters:

[in]	*UdpIo*	The pointer to the Udp6 Io.
[in]	*Context*	The pointer to the context.

Return values:

EFI_SUCCESS	This value is always returned.

Referenced by EfiMtftp6Configure(), and Mtftp6CreateService().

```
EFI_STATUS Mtftp6ConfigUdpIo ( IN UDP_IO *                UdpIo,
                               IN EFI_IPv6_ADDRESS *      ServerIp,
                               IN UINT16                  ServerPort,
                               IN EFI_IPv6_ADDRESS *      LocalIp,
                               IN UINT16                  LocalPort
                             )
```

The configure routine for Mtftp6 instance to transmit/receive.

Parameters:

[in]	*UdpIo*	The pointer to the Udp6 Io.
[in]	*ServerIp*	The pointer to the server address.
[in]	*ServerPort*	The pointer to the server port.
[in]	*LocalIp*	The pointer to the local address.
[in]	*LocalPort*	The pointer to the local port.

Return values:

EFI_SUCCESS	Configured the Udp6 Io for Mtftp6 successfully.
EFI_NO_MAPPING	The corresponding Ip6 instance has not been configured yet.

References Mtftp6GetMapping().

Referenced by Mtftp6TransmitPacket().

```
EFI_STATUS Mtftp6GetMapping ( IN UDP_IO *                  UdpIo,
                              IN EFI_UDP6_CONFIG_DATA *    UdpCfgData
                            )
```

Configure the opened Udp6 instance until the corresponding Ip6 instance has been configured.

Parameters:

[in]	*UdpIo*	The pointer to the Udp6 Io.
[in]	*UdpCfgData*	The pointer to the Udp6 configure data.

Return values:

EFI_SUCCESS	Configure the Udp6 instance successfully.
EFI_NO_MAPPING	The corresponding Ip6 instance has not been configured yet.

References MTFTP6_GET_MAPPING_TIMEOUT, and MTFTP6_TICK_PER_SECOND.

Referenced by Mtftp6ConfigUdpIo().

INTN Mtftp6GetNextBlockNum (IN LIST_ENTRY * *Head*)

Get the first valid block number on the range list.

Parameters:

[in] *Head* The block range head.

Return values:

==-1 If the block range is empty.

>-1 The first valid block number.

References MTFTP6_BLOCK_RANGE::Start.

Referenced by Mtftp6RrqHandleData(), Mtftp6RrqHandleOack(), Mtftp6WrqHandleAck(), and Mtftp6WrqHandleOack().

EFI_STATUS Mtftp6InitBlockRange (IN LIST_ENTRY * *Head*,
IN UINT16 *Start*,
IN UINT16 *End*
)

Initialize the block range for either RRQ or WRQ. RRQ and WRQ have different requirements for Start and End. For example, during startup, WRQ initializes its whole valid block range to [0, 0xffff]. This is bacause the server will send an ACK0 to inform the user to start the upload. When the client receives an ACK0, it will remove 0 from the range, get the next block number, which is 1, then upload the BLOCK1. For RRQ without option negotiation, the server will directly send the BLOCK1 in response to the client's RRQ. When received BLOCK1, the client will remove it from the block range and send an ACK. It also works if there is option negotiation.

Parameters:

[in] *Head* The block range head to initialize.

[in] *Start* The Start block number.

[in] *End* The last block number.

Return values:

EFI_OUT_OF_RESOURCES Failed to allocate memory for initial block range.

EFI_SUCCESS The initial block range is created.

References MTFTP6_BLOCK_RANGE::Link, and Mtftp6AllocateRange().

Referenced by Mtftp6RrqStart(), and Mtftp6WrqStart().

VOID EFIAPI Mtftp6OnPacketSent (IN NET_BUF * *Packet*,
IN UDP_END_POINT * *UdpEpt*,
IN EFI_STATUS *IoStatus*,
IN VOID * *Context*
)

The callback function called when the packet is transmitted.

Parameters:

[in] *Packet* The pointer to the packet.

[in] *UdpEpt* The pointer to the Udp6 access point.

[in] *IoStatus* The result of the transmission.
[in] *Context* The pointer to the context.

Referenced by Mtftp6TransmitPacket().

```
VOID EFIAPI Mtftp6OnTimerTick ( IN EFI_EVENT  Event,
                                IN VOID *     Context
                              )
```

The timer ticking routine for the Mtftp6 instance.

Parameters:

[in] *Event* The pointer to the ticking event.
[in] *Context* The pointer to the context.

References _MTFTP6_SERVICE::Children, _MTFTP6_INSTANCE::CurRetry, _MTFTP6_INSTANCE::LastPacket, _MTFTP6_INSTANCE::MaxRetry, _MTFTP6_INSTANCE::Mtftp6, Mtftp6OperationClean(), Mtftp6SendError(), Mtftp6TransmitPacket(), _MTFTP6_INSTANCE::PacketToLive, and _MTFTP6_INSTANCE::Token.

Referenced by Mtftp6CreateService().

```
VOID Mtftp6OperationClean ( IN MTFTP6_INSTANCE *  Instance,
                            IN EFI_STATUS         Result
                          )
```

Clean up the current Mtftp6 operation.

Parameters:

[in] *Instance* The pointer to the Mtftp6 instance.
[in] *Result* The result to be returned to the user.

Referenced by EfiMtftp6Configure(), Mtftp6OnTimerTick(), Mtftp6OperationStart(), Mtftp6RrqInput(), and Mtftp6WrqInput().

```
EFI_STATUS Mtftp6OperationStart ( IN EFI_MTFTP6_PROTOCOL *  This,
                                  IN EFI_MTFTP6_TOKEN *     Token,
                                  IN UINT16                 OpCode
                                )
```

Start the Mtftp6 instance to perform the operation, such as read file, write file, and read directory.

Parameters:

[in] *This* The MTFTP session.
[in] *Token* The token than encapsues the user's request.
[in] *OpCode* The operation to perform.

Return values:

EFI_INVALID_PARAMETER Some of the parameters are invalid.
EFI_NOT_STARTED The MTFTP session hasn't been configured.
EFI_ALREADY_STARTED There is pending operation for the session.

EFI_SUCCESS	The operation is successfully started.

References _MTFTP6_INSTANCE::BlkSize, _MTFTP6_INSTANCE::Config, _MTFTP6_INSTANCE::ExtInfo, _MTFTP6_INSTANCE::IsMaster, _MTFTP6_INSTANCE::MaxRetry, MTFTP6_DEFAULT_BLK_SIZE, MTFTP6_DEFAULT_MAX_RETRY, MTFTP6_DEFAULT_SERVER_CMD_PORT, MTFTP6_DEFAULT_TIMEOUT, MTFTP6_INSTANCE_FROM_THIS, Mtftp6OperationClean(), Mtftp6ParseExtensionOption(), Mtftp6RrqStart(), Mtftp6WrqStart(), _MTFTP6_INSTANCE::OldTpl, _MTFTP6_INSTANCE::ServerCmdPort, _MTFTP6_INSTANCE::ServerDataPort, _MTFTP6_INSTANCE::ServerIp, _MTFTP6_INSTANCE::Timeout, and _MTFTP6_INSTANCE::Token.

Referenced by EfiMtftp6GetInfo(), EfiMtftp6ReadDirectory(), EfiMtftp6ReadFile(), and EfiMtftp6WriteFile().

```
EFI_STATUS Mtftp6RemoveBlockNum ( IN LIST_ENTRY *  Head,
                                  IN UINT16        Num,
                                  IN BOOLEAN       Completed,
                                  OUT UINT64 *     TotalBlock
                                )
```

Remove the block number from the block range list.

Parameters:

[in]	*Head*	The block range list to remove from.
[in]	*Num*	The block number to remove.
[in]	*Completed*	Whether Num is the last block number
[out]	*TotalBlock*	The continuous block number in all

Return values:

EFI_NOT_FOUND	The block number isn't in the block range list.
EFI_SUCCESS	The block number has been removed from the list.
EFI_OUT_OF_RESOURCES	Failed to allocate resources.

References MTFTP6_BLOCK_RANGE::Bound, MTFTP6_BLOCK_RANGE::End, MTFTP6_BLOCK_RANGE::Link, Mtftp6AllocateRange(), MTFTP6_BLOCK_RANGE::Round, and MTFTP6_BLOCK_RANGE::Start.

Referenced by Mtftp6RrqSaveBlock(), and Mtftp6WrqHandleAck().

```
EFI_STATUS Mtftp6SendError ( IN MTFTP6_INSTANCE *  Instance,
                             IN UINT16             ErrCode,
                             IN UINT8 *            ErrInfo
                           )
```

Build and send an error packet.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*ErrCode*	The error code in the packet.
[in]	*ErrInfo*	The error message in the packet.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate memory for the error packet.
EFI_SUCCESS	The error packet is transmitted.
Others	Failed to transmit the packet.

References Mtftp6TransmitPacket().

Referenced by Mtftp6OnTimerTick(), Mtftp6RrqHandleOack(), Mtftp6RrqInput(), Mtftp6RrqSaveBlock(), Mtftp6WrqHandleAck(), Mtftp6WrqHandleOack(), Mtftp6WrqInput(), and Mtftp6WrqSendBlock().

```
EFI_STATUS Mtftp6SendRequest ( IN MTFTP6_INSTANCE *   Instance,
                               IN UINT16              Operation
                             )
```

Build and transmit the request packet for the Mtftp6 instance.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*Operation*	The operation code of this packet.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate memory for the request.
EFI_SUCCESS	The request is built and sent.
Others	Failed to transmit the packet.

References Mtftp6TransmitPacket().

Referenced by Mtftp6RrqStart(), and Mtftp6WrqStart().

```
VOID Mtftp6SetLastBlockNum ( IN LIST_ENTRY *   Head,
                             IN UINT16         Last
                           )
```

Set the last block number of the block range list. It removes all the blocks after the Last. MTFTP initialize the block range to the maximum possible range, such as [0, 0xffff] for WRQ. When it gets the last block number, it calls this function to set the last block number.

Parameters:

[in]	*Head*	The block range list.
[in]	*Last*	The last block number.

References MTFTP6_BLOCK_RANGE::End, MTFTP6_BLOCK_RANGE::Link, and MTFTP6_BLOCK_RANGE::Start.

Referenced by Mtftp6RrqSaveBlock(), and Mtftp6WrqSendBlock().

```
EFI_STATUS Mtftp6TransmitPacket ( IN MTFTP6_INSTANCE *   Instance,
                                  IN NET_BUF *           Packet
                                )
```

Send the packet for the Mtftp6 instance.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*Packet*	The pointer to the packet to be sent.

Return values:

EFI_SUCCESS	The packet was sent out
Others	Failed to transmit the packet.

References Mtftp6ConfigUdpIo(), and Mtftp6OnPacketSent().

Referenced by Mtftp6OnTimerTick(), Mtftp6RrqHandleData(), Mtftp6RrqSendAck(), Mtftp6SendError(), Mtftp6SendRequest(), and Mtftp6WrqSendBlock().

Generated on Mon Sep 28 08:49:56 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Mtftp6Dxe/Mtftp6Support.h File Reference

Data Structures

struct MTFTP6_BLOCK_RANGE

Functions

EFI_STATUS Mtftp6InitBlockRange (IN LIST_ENTRY *Head, IN UINT16 Start, IN UINT16 End)
INTN Mtftp6GetNextBlockNum (IN LIST_ENTRY *Head)
VOID Mtftp6SetLastBlockNum (IN LIST_ENTRY *Head, IN UINT16 Last)
EFI_STATUS Mtftp6RemoveBlockNum (IN LIST_ENTRY *Head, IN UINT16 Num, IN BOOLEAN Completed, OUT UINT64 *TotalBlock)
EFI_STATUS Mtftp6SendRequest (IN MTFTP6_INSTANCE *Instance, IN UINT16 Operation)
EFI_STATUS Mtftp6SendError (IN MTFTP6_INSTANCE *Instance, IN UINT16 ErrCode, IN UINT8 *ErrInfo)
EFI_STATUS Mtftp6TransmitPacket (IN MTFTP6_INSTANCE *Instance, IN NET_BUF *Packet)
EFI_STATUS EFIAPI Mtftp6CheckPacket (IN EFI_MTFTP6_PROTOCOL *This, IN EFI_MTFTP6_TOKEN *Token, IN UINT16 PacketLen, IN EFI_MTFTP6_PACKET *Packet)
EFI_STATUS EFIAPI Mtftp6ConfigDummyUdpIo (IN UDP_IO *UdpIo, IN VOID *Context)
EFI_STATUS Mtftp6ConfigUdpIo (IN UDP_IO *UdpIo, IN EFI_IPv6_ADDRESS *ServerIp, IN UINT16 ServerPort, IN EFI_IPv6_ADDRESS *LocalIp, IN UINT16 LocalPort)
VOID Mtftp6OperationClean (IN MTFTP6_INSTANCE *Instance, IN EFI_STATUS Result)
EFI_STATUS Mtftp6OperationStart (IN EFI_MTFTP6_PROTOCOL *This, IN EFI_MTFTP6_TOKEN *Token, IN UINT16 OpCode)
VOID EFIAPI Mtftp6OnTimerTick (IN EFI_EVENT Event, IN VOID *Context)
VOID EFIAPI Mtftp6WrqInput (IN NET_BUF *UdpPacket, IN UDP_END_POINT *UdpEpt, IN EFI_STATUS IoStatus, IN VOID *Context)
EFI_STATUS Mtftp6WrqStart (IN MTFTP6_INSTANCE *Instance, IN UINT16 Operation)
VOID EFIAPI Mtftp6RrqInput (IN NET_BUF *UdpPacket, IN UDP_END_POINT *UdpEpt, IN EFI_STATUS IoStatus, IN VOID *Context)
EFI_STATUS Mtftp6RrqStart (IN MTFTP6_INSTANCE *Instance, IN UINT16 Operation)

Detailed Description

Mtftp6 support functions declaration.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI Mtftp6CheckPacket ( IN EFI_MTFTP6_PROTOCOL *  This,
                                      IN EFI_MTFTP6_TOKEN *     Token,
                                      IN UINT16                 PacketLen,
                                      IN EFI_MTFTP6_PACKET *    Packet
                                    )
```

Check packet for GetInfo callback routine.

Parameters:

[in]	*This*	The pointer to the Mtftp6 protocol.
[in]	*Token*	The pointer to the Mtftp6 token.
[in]	*PacketLen*	The length of the packet
[in]	*Packet*	The pointer to the received packet.

Return values:

EFI_SUCCESS	The check process passed successfully.
EFI_ABORTED	Abort the Mtftp6 operation.

Check packet for GetInfo callback routine.

GetInfo is implemented with EfiMtftp6ReadFile. It's used to inspect the first packet from server, then abort the session.

Parameters:

[in]	*This*	The pointer to the Mtftp6 protocol.
[in]	*Token*	The pointer to the Mtftp6 token.
[in]	*PacketLen*	The length of the packet.
[in]	*Packet*	The pointer to the received packet.

Return values:

EFI_ABORTED	Abort the Mtftp6 operation.

References _MTFTP6_GETINFO_CONTEXT::Packet, _MTFTP6_GETINFO_CONTEXT::PacketLen, and _MTFTP6_GETINFO_CONTEXT::Status.

Referenced by EfiMtftp6GetInfo().

```
EFI_STATUS EFIAPI Mtftp6ConfigDummyUdpIo ( IN UDP_IO *  UdpIo,
                                           IN VOID *    Context
                                         )
```

The dummy configure routine for create a new Udp6 Io.

Parameters:

[in]	*UdpIo*	The pointer to the Udp6 Io.
[in]	*Context*	The pointer to the context.

Return values:

EFI_SUCCESS	The value is always returned.

The dummy configure routine for create a new Udp6 Io.

Parameters:

[in]	*UdpIo*	The pointer to the Udp6 Io.
[in]	*Context*	The pointer to the context.

Return values:

EFI_SUCCESS	This value is always returned.

Referenced by EfiMtftp6Configure(), and Mtftp6CreateService().

```
EFI_STATUS Mtftp6ConfigUdpIo ( IN UDP_IO *                UdpIo,
                               IN EFI_IPv6_ADDRESS *      ServerIp,
                               IN UINT16                  ServerPort,
                               IN EFI_IPv6_ADDRESS *      LocalIp,
                               IN UINT16                  LocalPort
                             )
```

The configure routine for the Mtftp6 instance to transmit/receive.

Parameters:

[in]	*UdpIo*	The pointer to the Udp6 Io.
[in]	*ServerIp*	The pointer to the server address.
[in]	*ServerPort*	The pointer to the server port.
[in]	*LocalIp*	The pointer to the local address.
[in]	*LocalPort*	The pointer to the local port.

Return values:

EFI_SUCCESS	Configure the Udp6 Io for Mtftp6 successfully.
EFI_NO_MAPPING	The corresponding Ip6 instance has not been configured yet.

The configure routine for Mtftp6 instance to transmit/receive.

Parameters:

[in]	*UdpIo*	The pointer to the Udp6 Io.
[in]	*ServerIp*	The pointer to the server address.
[in]	*ServerPort*	The pointer to the server port.
[in]	*LocalIp*	The pointer to the local address.
[in]	*LocalPort*	The pointer to the local port.

Return values:

EFI_SUCCESS	Configured the Udp6 Io for Mtftp6 successfully.
EFI_NO_MAPPING	The corresponding Ip6 instance has not been configured yet.

References Mtftp6GetMapping().

Referenced by Mtftp6TransmitPacket().

```
INTN Mtftp6GetNextBlockNum ( IN LIST_ENTRY *   Head   )
```

Get the first valid block number on the range list.

Parameters:

[in] *Head* The block range head.

Return values:

==-1 If the block range is empty.

>-1 The first valid block number.

References MTFTP6_BLOCK_RANGE::Start.

Referenced by Mtftp6RrqHandleData(), Mtftp6RrqHandleOack(), Mtftp6WrqHandleAck(), and Mtftp6WrqHandleOack().

```
EFI_STATUS Mtftp6InitBlockRange ( IN LIST_ENTRY *  Head,
                                  IN UINT16        Start,
                                  IN UINT16        End
                                )
```

Initialize the block range for either RRQ or WRQ. RRQ and WRQ have different requirements for Start and End. For example, during startup, WRQ initializes its whole valid block range to [0, 0xffff]. This is because the server will send an ACK0 to inform the user to start the upload. When the client receives an ACK0, it will remove 0 from the range, get the next block number, which is 1, then upload the BLOCK1. For RRQ without option negotiation, the server will directly send us the BLOCK1 in response to the client's RRQ. When BLOCK1 is received, the client will remove it from the block range and send an ACK. It also works if there is option negotiation.

Parameters:

[in] *Head* The block range head to initialize.

[in] *Start* The Start block number.

[in] *End* The last block number.

Return values:

EFI_OUT_OF_RESOURCES Failed to allocate memory for initial block range.

EFI_SUCCESS The initial block range is created.

Initialize the block range for either RRQ or WRQ. RRQ and WRQ have different requirements for Start and End. For example, during startup, WRQ initializes its whole valid block range to [0, 0xffff]. This is bacause the server will send an ACK0 to inform the user to start the upload. When the client receives an ACK0, it will remove 0 from the range, get the next block number, which is 1, then upload the BLOCK1. For RRQ without option negotiation, the server will directly send the BLOCK1 in response to the client's RRQ. When received BLOCK1, the client will remove it from the block range and send an ACK. It also works if there is option negotiation.

Parameters:

[in] *Head* The block range head to initialize.

[in] *Start* The Start block number.

[in] *End* The last block number.

Return values:

EFI_OUT_OF_RESOURCES Failed to allocate memory for initial block range.

EFI_SUCCESS The initial block range is created.

References MTFTP6_BLOCK_RANGE::Link, and Mtftp6AllocateRange().

Referenced by Mtftp6RrqStart(), and Mtftp6WrqStart().

```
VOID EFIAPI Mtftp6OnTimerTick ( IN EFI_EVENT  Event,
                                IN VOID *     Context
                              )
```

The timer ticking routine for the Mtftp6 instance.

Parameters:

[in]	*Event*	The pointer to the ticking event.
[in]	*Context*	The pointer to the context.

References _MTFTP6_SERVICE::Children, _MTFTP6_INSTANCE::CurRetry, _MTFTP6_INSTANCE::LastPacket, _MTFTP6_INSTANCE::MaxRetry, _MTFTP6_INSTANCE::Mtftp6, Mtftp6OperationClean(), Mtftp6SendError(), Mtftp6TransmitPacket(), _MTFTP6_INSTANCE::PacketToLive, and _MTFTP6_INSTANCE::Token.

Referenced by Mtftp6CreateService().

```
VOID Mtftp6OperationClean ( IN MTFTP6_INSTANCE *  Instance,
                            IN EFI_STATUS         Result
                          )
```

Clean up the current Mtftp6 operation.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*Result*	The result to be returned to the user.

Referenced by EfiMtftp6Configure(), Mtftp6OnTimerTick(), Mtftp6OperationStart(), Mtftp6RrqInput(), and Mtftp6WrqInput().

```
EFI_STATUS Mtftp6OperationStart ( IN EFI_MTFTP6_PROTOCOL *  This,
                                  IN EFI_MTFTP6_TOKEN *     Token,
                                  IN UINT16                 OpCode
                                )
```

Start the Mtftp6 instance to perform the operation, such as read file, write file, and read directory.

Parameters:

[in]	*This*	The MTFTP session
[in]	*Token*	The token that encapsulates the user's request.
[in]	*OpCode*	The operation to perform.

Return values:

EFI_INVALID_PARAMETER	Some of the parameters are invalid.
EFI_NOT_STARTED	The MTFTP session hasn't been configured.
EFI_ALREADY_STARTED	There is pending operation for the session.
EFI_SUCCESS	The operation was successfully started.

Start the Mtftp6 instance to perform the operation, such as read file, write file, and read directory.

Parameters:

[in]	*This*	The MTFTP session.
[in]	*Token*	The token than encapsues the user's request.
[in]	*OpCode*	The operation to perform.

Return values:

EFI_INVALID_PARAMETER	Some of the parameters are invalid.
EFI_NOT_STARTED	The MTFTP session hasn't been configured.
EFI_ALREADY_STARTED	There is pending operation for the session.
EFI_SUCCESS	The operation is successfully started.

References _MTFTP6_INSTANCE::BlkSize, _MTFTP6_INSTANCE::Config, _MTFTP6_INSTANCE::ExtInfo, _MTFTP6_INSTANCE::IsMaster, _MTFTP6_INSTANCE::MaxRetry, MTFTP6_DEFAULT_BLK_SIZE, MTFTP6_DEFAULT_MAX_RETRY, MTFTP6_DEFAULT_SERVER_CMD_PORT, MTFTP6_DEFAULT_TIMEOUT, MTFTP6_INSTANCE_FROM_THIS, Mtftp6OperationClean(), Mtftp6ParseExtensionOption(), Mtftp6RrqStart(), Mtftp6WrqStart(), _MTFTP6_INSTANCE::OldTpl, _MTFTP6_INSTANCE::ServerCmdPort, _MTFTP6_INSTANCE::ServerDataPort, _MTFTP6_INSTANCE::ServerIp, _MTFTP6_INSTANCE::Timeout, and _MTFTP6_INSTANCE::Token.

Referenced by EfiMtftp6GetInfo(), EfiMtftp6ReadDirectory(), EfiMtftp6ReadFile(), and EfiMtftp6WriteFile().

```
EFI_STATUS Mtftp6RemoveBlockNum ( IN LIST_ENTRY *  Head,
                                  IN UINT16        Num,
                                  IN BOOLEAN       Completed,
                                  OUT UINT64 *     TotalBlock
                                )
```

Remove the block number from the block range list.

Parameters:

[in]	*Head*	The block range list to remove from.
[in]	*Num*	The block number to remove.
[in]	*Completed*	Whether Num is the last block number
[out]	*TotalBlock*	The continuous block number in all

Return values:

EFI_NOT_FOUND	The block number isn't in the block range list.
EFI_SUCCESS	The block number has been removed from the list.
EFI_OUT_OF_RESOURCES	Failed to allocate resources.

References MTFTP6_BLOCK_RANGE::Bound, MTFTP6_BLOCK_RANGE::End, MTFTP6_BLOCK_RANGE::Link, Mtftp6AllocateRange(), MTFTP6_BLOCK_RANGE::Round, and MTFTP6_BLOCK_RANGE::Start.

Referenced by Mtftp6RrqSaveBlock(), and Mtftp6WrqHandleAck().

```
VOID EFIAPI Mtftp6RrqInput ( IN NET_BUF *          UdpPacket,
                             IN UDP_END_POINT *    UdpEpt,
                             IN EFI_STATUS         IoStatus,
```

IN VOID * *Context*
)

The packet process callback for Mtftp6 download.

Parameters:

[in]	*UdpPacket*	The pointer to the packet received.
[in]	*UdpEpt*	The pointer to the Udp6 access point.
[in]	*IoStatus*	The status from Udp6 instance.
[in]	*Context*	The pointer to the context.

References _MTFTP6_INSTANCE::BlkSize, _MTFTP6_INSTANCE::McastIp, _MTFTP6_INSTANCE::McastUdpIo, _MTFTP6_INSTANCE::Mtftp6, MTFTP6_DATA_HEAD_LEN, MTFTP6_INSTANCE_SIGNATURE, MTFTP6_OPCODE_LEN, Mtftp6OperationClean(), Mtftp6RrqHandleData(), Mtftp6RrqHandleOack(), Mtftp6RrqInput(), Mtftp6SendError(), _MTFTP6_INSTANCE::ServerDataPort, _MTFTP6_INSTANCE::Token, and _MTFTP6_INSTANCE::UdpIo.

Referenced by Mtftp6RrqHandleOack(), Mtftp6RrqInput(), and Mtftp6RrqStart().

EFI_STATUS Mtftp6RrqStart (IN MTFTP6_INSTANCE * *Instance*,
IN UINT16 *Operation*
)

Start the Mtftp6 instance to download. It first initializes some of the internal states then builds and sends an RRQ reqeuest packet. Finally, it starts receive for the downloading.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*Operation*	The operation code of current packet.

Return values:

EFI_SUCCESS	The Mtftp6 was started to download.
Others	Failed to start to download.

Start the Mtftp6 instance to download. It first initializes some of the internal states, then builds and sends an RRQ reqeuest packet. Finally, it starts receive for the downloading.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*Operation*	The operation code of current packet.

Return values:

EFI_SUCCESS	The Mtftp6 is started to download.
Others	Failed to start to download.

References Mtftp6InitBlockRange(), Mtftp6RrqInput(), and Mtftp6SendRequest().

Referenced by Mtftp6OperationStart().

EFI_STATUS Mtftp6SendError (IN MTFTP6_INSTANCE * *Instance*,
IN UINT16 *ErrCode*,

IN UINT8 * *ErrInfo*
)

Build and send an error packet.

Parameters:

[in] *Instance* The pointer to the Mtftp6 instance.
[in] *ErrCode* The error code in the packet.
[in] *ErrInfo* The error message in the packet.

Return values:

EFI_OUT_OF_RESOURCES Failed to allocate memory for the error packet.
EFI_SUCCESS The error packet was transmitted.
Others Failed to transmit the packet.

Build and send an error packet.

Parameters:

[in] *Instance* The pointer to the Mtftp6 instance.
[in] *ErrCode* The error code in the packet.
[in] *ErrInfo* The error message in the packet.

Return values:

EFI_OUT_OF_RESOURCES Failed to allocate memory for the error packet.
EFI_SUCCESS The error packet is transmitted.
Others Failed to transmit the packet.

References Mtftp6TransmitPacket().

Referenced by Mtftp6OnTimerTick(), Mtftp6RrqHandleOack(), Mtftp6RrqInput(), Mtftp6RrqSaveBlock(), Mtftp6WrqHandleAck(), Mtftp6WrqHandleOack(), Mtftp6WrqInput(), and Mtftp6WrqSendBlock().

EFI_STATUS Mtftp6SendRequest (IN MTFTP6_INSTANCE * *Instance*,
IN UINT16 *Operation*
)

Build and transmit the request packet for the Mtftp6 instance.

Parameters:

[in] *Instance* The pointer to the Mtftp6 instance.
[in] *Operation* The operation code of this packet.

Return values:

EFI_OUT_OF_RESOURCES Failed to allocate memory for the request.
EFI_SUCCESS The request was built and sent.
Others Failed to transmit the packet.

Build and transmit the request packet for the Mtftp6 instance.

Parameters:

[in] *Instance* The pointer to the Mtftp6 instance.
[in] *Operation* The operation code of this packet.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate memory for the request.
EFI_SUCCESS	The request is built and sent.
Others	Failed to transmit the packet.

References Mtftp6TransmitPacket().

Referenced by Mtftp6RrqStart(), and Mtftp6WrqStart().

```
VOID Mtftp6SetLastBlockNum ( IN LIST_ENTRY *  Head,
                             IN UINT16        Last
                           )
```

Set the last block number of the block range list. It removes all the blocks after the Last. MTFTP initialize the block range to the maximum possible range, such as [0, 0xffff] for WRQ. When it gets the last block number, it calls this function to set the last block number.

Parameters:

[in]	*Head*	The block range list.
[in]	*Last*	The last block number.

References MTFTP6_BLOCK_RANGE::End, MTFTP6_BLOCK_RANGE::Link, and MTFTP6_BLOCK_RANGE::Start.

Referenced by Mtftp6RrqSaveBlock(), and Mtftp6WrqSendBlock().

```
EFI_STATUS Mtftp6TransmitPacket ( IN MTFTP6_INSTANCE *  Instance,
                                  IN NET_BUF *          Packet
                                )
```

Send the packet for the Mtftp6 instance.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*Packet*	The pointer to the packet to be sent.

Return values:

EFI_SUCCESS	The packet was sent out
Others	Failed to transmit the packet.

References Mtftp6ConfigUdpIo(), and Mtftp6OnPacketSent().

Referenced by Mtftp6OnTimerTick(), Mtftp6RrqHandleData(), Mtftp6RrqSendAck(), Mtftp6SendError(), Mtftp6SendRequest(), and Mtftp6WrqSendBlock().

```
VOID EFIAPI Mtftp6WrqInput ( IN NET_BUF *          UdpPacket,
                             IN UDP_END_POINT *    UdpEpt,
                             IN EFI_STATUS         IoStatus,
                             IN VOID *             Context
                           )
```

The packet process callback for Mtftp6 upload.

Parameters:

[in]	*UdpPacket*	The pointer to the packet received.
[in]	*UdpEpt*	The pointer to the Udp6 access point.
[in]	*IoStatus*	The status from the Udp6 instance.
[in]	*Context*	The pointer to the context.

The packet process callback for Mtftp6 upload.

Parameters:

[in]	*UdpPacket*	The pointer to the packet received.
[in]	*UdpEpt*	The pointer to the Udp6 access point.
[in]	*IoStatus*	The status from Udp6 instance.
[in]	*Context*	The pointer to the context.

References _MTFTP6_INSTANCE::Mtftp6, MTFTP6_BLKNO_LEN, MTFTP6_INSTANCE_SIGNATURE, MTFTP6_OPCODE_LEN, Mtftp6OperationClean(), Mtftp6SendError(), Mtftp6WrqHandleAck(), Mtftp6WrqHandleOack(), Mtftp6WrqInput(), _MTFTP6_INSTANCE::ServerDataPort, _MTFTP6_INSTANCE::Token, and _MTFTP6_INSTANCE::UdpIo.

Referenced by Mtftp6WrqInput(), and Mtftp6WrqStart().

```
EFI_STATUS Mtftp6WrqStart ( IN MTFTP6_INSTANCE *   Instance,
                            IN UINT16              Operation
                          )
```

Start the Mtftp6 instance to upload. It will first init some states, then send the WRQ request packet, and start to receive the packet.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*Operation*	The operation code of current packet.

Return values:

EFI_SUCCESS	The Mtftp6 was started to upload.
Others	Failed to start to upload.

Start the Mtftp6 instance to upload. It will first init some states, then send the WRQ request packet, and start to receive the packet.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*Operation*	The operation code of the current packet.

Return values:

EFI_SUCCESS	The Mtftp6 was started to upload.
Others	Failed to start to upload.

References Mtftp6InitBlockRange(), Mtftp6SendRequest(), and Mtftp6WrqInput().

Referenced by Mtftp6OperationStart().

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Mtftp6Dxe/Mtftp6Wrq.c File Reference

Functions

EFI_STATUS Mtftp6WrqSendBlock (IN MTFTP6_INSTANCE *Instance, IN UINT16 BlockNum)
EFI_STATUS Mtftp6WrqHandleAck (IN MTFTP6_INSTANCE *Instance, IN EFI_MTFTP6_PACKET *Packet, IN UINT32 Len, OUT NET_BUF **UdpPacket, OUT BOOLEAN *IsCompleted)
BOOLEAN Mtftp6WrqOackValid (IN MTFTP6_EXT_OPTION_INFO *ReplyInfo, IN MTFTP6_EXT_OPTION_INFO *RequestInfo)
EFI_STATUS Mtftp6WrqHandleOack (IN MTFTP6_INSTANCE *Instance, IN EFI_MTFTP6_PACKET *Packet, IN UINT32 Len, OUT NET_BUF **UdpPacket, OUT BOOLEAN *IsCompleted)
VOID EFIAPI Mtftp6WrqInput (IN NET_BUF *UdpPacket, IN UDP_END_POINT *UdpEpt, IN EFI_STATUS IoStatus, IN VOID *Context)
EFI_STATUS Mtftp6WrqStart (IN MTFTP6_INSTANCE *Instance, IN UINT16 Operation)

Detailed Description

Mtftp6 Wrq process functions implementation.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS Mtftp6WrqHandleAck ( IN MTFTP6_INSTANCE *        Instance,
                                IN EFI_MTFTP6_PACKET *      Packet,
                                IN UINT32                   Len,
                                OUT NET_BUF **              UdpPacket,
                                OUT BOOLEAN *               IsCompleted
                              )
```

Function to handle received ACK packet. If the ACK number matches the expected block number, with more data pending, send the next block. Otherwise, tell the caller that we are done.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*Packet*	The pointer to the received packet.
[in]	*Len*	The length of the packet.
[out]	*UdpPacket*	The net buf of received packet.
[out]	*IsCompleted*	If TRUE, the upload has been completed. Otherwise, the upload has not been completed.

Return values:

EFI_SUCCESS	The ACK packet successfully processed.
EFI_TFTP_ERROR	The block number loops back.
Others	Failed to transmit the next data packet.

References Mtftp6GetNextBlockNum(), Mtftp6RemoveBlockNum(), Mtftp6SendError(), and Mtftp6WrqSendBlock().

Referenced by Mtftp6WrqHandleOack(), and Mtftp6WrqInput().

```
EFI_STATUS Mtftp6WrqHandleOack ( IN MTFTP6_INSTANCE *      Instance,
                                 IN EFI_MTFTP6_PACKET *    Packet,
                                 IN UINT32                 Len,
                                 OUT NET_BUF **            UdpPacket,
                                 OUT BOOLEAN *             IsCompleted
                               )
```

Process the OACK packet for Wrq.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*Packet*	The pointer to the received packet.
[in]	*Len*	The length of the packet.
[out]	*UdpPacket*	The net buf of received packet.
[out]	*IsCompleted*	If TRUE, the upload has been completed. Otherwise, the upload has not been completed.

Return values:

EFI_SUCCESS	The OACK packet successfully processed.
EFI_TFTP_ERROR	An TFTP communication error happened.
Others	Failed to process the OACK packet.

References MTFTP6_EXT_OPTION_INFO::BlkSize, Mtftp6GetNextBlockNum(), Mtftp6ParseExtensionOption(), Mtftp6ParseStart(), Mtftp6SendError(), Mtftp6WrqHandleAck(), Mtftp6WrqOackValid(), and MTFTP6_EXT_OPTION_INFO::Timeout.

Referenced by Mtftp6WrqInput().

```
VOID EFIAPI Mtftp6WrqInput ( IN NET_BUF *          UdpPacket,
                             IN UDP_END_POINT *    UdpEpt,
                             IN EFI_STATUS         IoStatus,
                             IN VOID *             Context
                           )
```

The packet process callback for Mtftp6 upload.

Parameters:

[in]	*UdpPacket*	The pointer to the packet received.
[in]	*UdpEpt*	The pointer to the Udp6 access point.
[in]	*IoStatus*	The status from Udp6 instance.
[in]	*Context*	The pointer to the context.

References _MTFTP6_INSTANCE::Mtftp6, MTFTP6_BLKNO_LEN, MTFTP6_INSTANCE_SIGNATURE, MTFTP6_OPCODE_LEN, Mtftp6OperationClean(), Mtftp6SendError(), Mtftp6WrqHandleAck(), Mtftp6WrqHandleOack(), Mtftp6WrqInput(), _MTFTP6_INSTANCE::ServerDataPort, _MTFTP6_INSTANCE::Token, and _MTFTP6_INSTANCE::UdpIo.

Referenced by Mtftp6WrqInput(), and Mtftp6WrqStart().

```
BOOLEAN Mtftp6WrqOackValid ( IN MTFTP6_EXT_OPTION_INFO *  ReplyInfo,
                             IN MTFTP6_EXT_OPTION_INFO *  RequestInfo
                           )
```

Check whether the received OACK is valid. The OACK is valid only if: 1. It only include options requested by us. 2. It can only include a smaller block size. 3. It can't change the proposed time out value. 4. Other requirements of the individual MTFTP6 options as required.

Parameters:

[in] *ReplyInfo* The pointer to options information in reply packet.
[in] *RequestInfo* The pointer to requested options information.

Return values:

TRUE If the option in OACK is valid.
FALSE If the option is invalid.

References MTFTP6_OPT_BLKSIZE_BIT, and MTFTP6_OPT_TIMEOUT_BIT.

Referenced by Mtftp6WrqHandleOack().

```
EFI_STATUS Mtftp6WrqSendBlock ( IN MTFTP6_INSTANCE *  Instance,
                                IN UINT16            BlockNum
                              )
```

Build and send a Mtftp6 data packet for upload.

Parameters:

[in] *Instance* The pointer to the Mtftp6 instance.
[in] *BlockNum* The block num to be sent.

Return values:

EFI_OUT_OF_RESOURCES Failed to allocate memory for the packet.
EFI_SUCCESS The data packet was sent.
EFI_ABORTED The user aborted this process.

References MTFTP6_DATA_HEAD_LEN, Mtftp6SendError(), Mtftp6SetLastBlockNum(), and Mtftp6TransmitPacket().

Referenced by Mtftp6WrqHandleAck().

```
EFI_STATUS Mtftp6WrqStart ( IN MTFTP6_INSTANCE *  Instance,
                            IN UINT16            Operation
                          )
```

Start the Mtftp6 instance to upload. It will first init some states, then send the WRQ request packet, and start to receive the packet.

Parameters:

[in]	*Instance*	The pointer to the Mtftp6 instance.
[in]	*Operation*	The operation code of the current packet.

Return values:

EFI_SUCCESS	The Mtftp6 was started to upload.
Others	Failed to start to upload.

References Mtftp6InitBlockRange(), Mtftp6SendRequest(), and Mtftp6WrqInput().

Referenced by Mtftp6OperationStart().

Generated on Mon Sep 28 08:49:56 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/TcpDxe/ComponentName.c File Reference

Functions

EFI_STATUS EFIAPI TcpComponentNameGetDriverName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN CHAR8 *Language, OUT CHAR16 **DriverName)

EFI_STATUS EFIAPI TcpComponentNameGetControllerName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE ChildHandle, IN CHAR8 *Language, OUT CHAR16 **ControllerName)

EFI_STATUS UpdateTcp4Name (IN EFI_TCP4_PROTOCOL *Tcp4)

EFI_STATUS UpdateTcp6Name (IN EFI_TCP6_PROTOCOL *Tcp6)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gTcpComponentName

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gTcpComponentName2

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mTcpDriverNameTable []

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE * gTcpControllerNameTable = NULL

Detailed Description

Implementation of protocols EFI_COMPONENT_NAME_PROTOCOL and EFI_COMPONENT_NAME2_PROTOCOL.

Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI TcpComponentNameGetControllerName (IN EFI_COMPONENT_NAME_PROTOCOL * *This*,
IN EFI_HANDLE *ControllerHandle*,

```
                                IN EFI_HANDLE                    ChildHandle,
                                IN CHAR8 *                       Language,
                                OUT CHAR16 **                    ControllerName
                              )
```

Retrieves a Unicode string that is the user-readable name of the controller that is being managed by a driver.

This function retrieves the user-readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
[in]	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language, from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL, and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language or ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gTcpComponentName, gTcpControllerNameTable, UpdateTcp4Name(), and UpdateTcp6Name().

```
EFI_STATUS EFIAPI
TcpComponentNameGetDriverName   ( IN
                                  EFI_COMPONENT_NAME_PROTOCOL *   This,
                                  IN CHAR8 *                      Language,
                                  OUT CHAR16 **                   DriverName
```

)

Retrieves a Unicode string that is the user-readable name of the driver.

This function retrieves the user-readable name of a driver in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the driver name is returned in DriverName, and EFI_SUCCESS is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*DriverName*	A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by This in the language specified by Language.

Return values:

EFI_SUCCESS	The Unicode string for the Driver specified by This, and the language specified by Language was returned in DriverName.
EFI_INVALID_PARAMETER	Language or DriverName is NULL.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gTcpComponentName, and mTcpDriverNameTable.

EFI_STATUS UpdateTcp4Name (IN EFI_TCP4_PROTOCOL * *Tcp4*)

Update the component name for the Tcp4 child handle.

Parameters:

Tcp4[in] A pointer to the EFI_TCP4_PROTOCOL.

Return values:

EFI_SUCCESS	Update the ControllerNameTable of this instance successfully.
EFI_INVALID_PARAMETER	The input parameter is invalid.

References gTcpComponentName, gTcpComponentName2, and gTcpControllerNameTable.

Referenced by TcpComponentNameGetControllerName().

EFI_STATUS UpdateTcp6Name (IN EFI_TCP6_PROTOCOL * *Tcp6*)

Update the component name for the Tcp6 child handle.

Parameters:

Tcp6[in] A pointer to the EFI_TCP6_PROTOCOL.

Return values:

EFI_SUCCESS	Update the ControllerNameTable of this instance successfully.
EFI_INVALID_PARAMETER	The input parameter is invalid.

References gTcpComponentName, gTcpComponentName2, and gTcpControllerNameTable.

Referenced by TcpComponentNameGetControllerName().

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gTcpComponentName

Initial value:

```
{
 TcpComponentNameGetDriverName,
 TcpComponentNameGetControllerName,
 "eng"
}
```

EFI Component Name Protocol

Referenced by TcpComponentNameGetControllerName(), TcpComponentNameGetDriverName(), TcpDriverEntryPoint(), UpdateTcp4Name(), and UpdateTcp6Name().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gTcpComponentName2

Initial value:

```
{
 (EFI_COMPONENT_NAME2_GET_DRIVER_NAME) TcpComponentNameGetDriverName,
 (EFI_COMPONENT_NAME2_GET_CONTROLLER_NAME) TcpComponentNameGetControllerName,
 "en"
}
```

EFI Component Name 2 Protocol

Referenced by TcpDriverEntryPoint(), UpdateTcp4Name(), and UpdateTcp6Name().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE* gTcpControllerNameTable = NULL

Referenced by TcpComponentNameGetControllerName(), UpdateTcp4Name(), and UpdateTcp6Name().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mTcpDriverNameTable[]

Initial value:

```
{
 {
   "eng;en",
   L"TCP Network Service Driver"
 },
 {
   NULL,
   NULL
```

```
  }
}
```

Referenced by TcpComponentNameGetDriverName().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/TcpDxe/Socket.h File Reference

Data Structures

struct _SOCK_COMPLETION_TOKEN
union SOCK_IO_DATA
struct _SOCK_IO_TOKEN
struct _SOCK_BUFFER
struct _SOCK_INIT_DATA
union _NET_PROTOCOL
struct _TCP_SOCKET
struct _SOCK_TOKEN
struct _TCP_RSV_DATA

Defines

"memItemRight" valign="bottom">SOCK_SND_BUF 0

#define SOCK_RCV_BUF 1
#define SOCK_BUFF_LOW_WATER (2 * 1024)
#define SOCK_RCV_BUFF_SIZE (8 * 1024)
#define SOCK_SND_BUFF_SIZE (8 * 1024)
#define SOCK_BACKLOG 5
#define PROTO_RESERVED_LEN 20
#define SO_NO_MORE_DATA 0x0001
#define SO_CLOSED 0
#define SO_LISTENING 1
#define SO_CONNECTING 2
#define SO_CONNECTED 3
#define SO_DISCONNECTING 4
#define SO_UNCONFIGURED 0
#define SO_CONFIGURED_ACTIVE 1
#define SO_CONFIGURED_PASSIVE 2
#define SO_NO_MAPPING 3
#define SOCK_ATTACH 0
Attach current socket to a new PCB.
#define SOCK_DETACH 1
Detach current socket from the PCB.
#define SOCK_CONFIGURE 2
Configure attached PCB.
#define SOCK_FLUSH 3
Flush attached PCB.
#define SOCK_SND 4
Need protocol to send something.
#define SOCK_SNDPUSH 5
Need protocol to send pushed data.
#define SOCK_SNDURG 6
Need protocol to send urgent data.
#define SOCK_CONSUMED 7
Application has retrieved data from socket.
#define SOCK_CONNECT 8
Need to connect to a peer.
#define SOCK_CLOSE 9

Need to close the protocol process.

#define SOCK_ABORT 10

Need to reset the protocol process.

#define SOCK_POLL 11

Need to poll to the protocol layer.

#define SOCK_ROUTE 12

Need to add a route information.

#define SOCK_MODE 13

Need to get the mode data of the protocol.

#define SOCK_GROUP 14

Need to join a mcast group.

#define SOCK_NO_MORE_DATA(Sock) ((Sock)->Flag |= SO_NO_MORE_DATA)

#define SOCK_IS_UNCONFIGURED(Sock) ((Sock)->ConfigureState == SO_UNCONFIGURED)

#define SOCK_IS_CONFIGURED(Sock)

#define SOCK_IS_CONFIGURED_ACTIVE(Sock) ((Sock)->ConfigureSt SO_CONFIGURED_ACTIVE)

#define SOCK_IS_CONNECTED_PASSIVE(Sock) ((Sock)->ConfigureSt SO_CONFIGURED_PASSIVE)

#define SOCK_IS_NO_MAPPING(Sock) ((Sock)->ConfigureState == SO_NO_MAPPING)

#define SOCK_IS_CLOSED(Sock) ((Sock)->State == SO_CLOSED)

#define SOCK_IS_LISTENING(Sock) ((Sock)->State == SO_LISTENIN

#define SOCK_IS_CONNECTING(Sock) ((Sock)->State == SO_CONNE

#define SOCK_IS_CONNECTED(Sock) ((Sock)->State == SO_CONNEC

#define SOCK_IS_DISCONNECTING(Sock) ((Sock)->State == SO_DISCONNECTING)

#define SOCK_IS_NO_MORE_DATA(Sock) (0 != ((Sock)->Flag & SO_NO_MORE_DATA))

#define SET_RCV_BUFFSIZE(Sock, Size) ((Sock)->RcvBuffer.HighWat (Size))

#define GET_RCV_BUFFSIZE(Sock) ((Sock)->RcvBuffer.HighWater)

#define GET_RCV_DATASIZE(Sock) (((Sock)->RcvBuffer.DataQueue)-

#define SET_SND_BUFFSIZE(Sock, Size) ((Sock)->SndBuffer.HighWat (Size))

#define GET_SND_BUFFSIZE(Sock) ((Sock)->SndBuffer.HighWater)

#define GET_SND_DATASIZE(Sock) (((Sock)->SndBuffer.DataQueue)-

#define SET_BACKLOG(Sock, Value) ((Sock)->BackLog = (Value))

#define GET_BACKLOG(Sock) ((Sock)->BackLog)

#define SOCK_ERROR(Sock, Error) ((Sock)->SockError = (Error))

#define SOCK_SIGNATURE SIGNATURE_32 ('S', 'O', 'C', 'K')

#define SOCK_FROM_THIS(a) CR ((a), SOCKET, NetProtocol, SOCK_SIGNATURE)

#define SOCK_FROM_TOKEN(Token) (((SOCK_TOKEN *) (Token))->

#define PROTO_TOKEN_FORM_SOCK(SockToken, Type) ((Type *) (((SOCK_TOKEN *) (SockToken))->Token))

Typedefs

typedef struct _TCP_SOCKET SOCKET

typedef struct _SOCK_COMPLETION_TOKEN SOCK_COMPLETION_TOKEN

typedef struct _SOCK_IO_TOKEN SOCK_IO_TOKEN

typedef struct _SOCK_BUFFER SOCK_BUFFER
typedef EFI_STATUS(* SOCK_PROTO_HANDLER)(IN SOCKET *Socket, IN U
VOID *RequestData)
typedef EFI_STATUS(* SOCK_CREATE_CALLBACK)(IN SOCKET *This, IN V
typedef VOID(* SOCK_DESTROY_CALLBACK)(IN SOCKET *This, IN
typedef struct _SOCK_INIT_DATA SOCK_INIT_DATA
typedef union _NET_PROTOCOL NET_PROTOCOL
typedef struct _SOCK_TOKEN SOCK_TOKEN
typedef struct _TCP_RSV_DATA TCP_RSV_DATA

Enumerations

enum SOCK_TYPE { SockDgram, SockStream }

Functions

VOID SockSetState (IN OUT SOCKET *Sock, IN UINT8 State)
SOCKET * SockClone (IN SOCKET *Sock)
VOID SockConnEstablished (IN OUT SOCKET *Sock)
VOID SockConnClosed (IN OUT SOCKET *Sock)
VOID SockDataSent (IN OUT SOCKET *Sock, IN UINT32 Cour
UINT32 SockGetDataToSend (IN SOCKET *Sock, IN UINT32 Off
Len, OUT UINT8 *Dest)
VOID SockDataRcvd (IN OUT SOCKET *Sock, IN OUT NET_B
IN UINT32 UrgLen)
UINT32 SockGetFreeSpace (IN SOCKET *Sock, IN UINT32 Which
VOID SockNoMoreData (IN OUT SOCKET *Sock)
SOCKET * SockCreateChild (IN SOCK_INIT_DATA *SockInitData)
EFI_STATUS SockDestroyChild (IN OUT SOCKET *Sock)
EFI_STATUS SockConfigure (IN SOCKET *Sock, IN VOID *ConfigDat
EFI_STATUS SockConnect (IN SOCKET *Sock, IN VOID *Token)
EFI_STATUS SockAccept (IN SOCKET *Sock, IN VOID *Token)
EFI_STATUS SockSend (IN SOCKET *Sock, IN VOID *Token)
EFI_STATUS SockRcv (IN SOCKET *Sock, IN VOID *Token)
EFI_STATUS SockFlush (IN OUT SOCKET *Sock)
EFI_STATUS SockClose (IN OUT SOCKET *Sock, IN VOID *Token, IN
OnAbort)
EFI_STATUS SockGetMode (IN SOCKET *Sock, IN OUT VOID *Mode
EFI_STATUS SockGroup (IN SOCKET *Sock, IN VOID *GroupInfo)
EFI_STATUS SockRoute (IN SOCKET *Sock, IN VOID *RouteInfo)

Detailed Description

Common head file for TCP socket.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"paramtype">Sock) ((Sock)->BackLog)

Get the backlog value of the socket.

Parameters:

`[in]` *Sock* Pointer to the socket.

Returns:

The backlog value.

Referenced by Tcp4GetMode(), and Tcp6GetMode().

#define GET_RCV_BUFFSIZE (Sock) ((Sock)->RcvBuffer.HighWater)

Get the size of the receive buffer.

Parameters:

`[in]` *Sock* Pointer to the socket.

Returns:

The receive buffer size.

Referenced by Tcp4GetMode(), Tcp6GetMode(), TcpComputeScale(), TcpComputeWnd(), TcpInitTcbLocal(), and TcpRcvWinNow().

#define GET_RCV_DATASIZE (Sock) (((Sock)->RcvBuffer.DataQueue)->BufSize)

Get the size of the receive data.

Parameters:

`[in]` *Sock* Pointer to the socket.

Returns:

The received data size.

Referenced by SockNoMoreData(), SockRcv(), and TcpOnAppClose().

#define GET_SND_BUFFSIZE (Sock) ((Sock)->SndBuffer.HighWater)

Get the size of the send buffer.

Parameters:

`[in]` *Sock* Pointer to the socket.

Returns:

The send buffer size.

Referenced by Tcp4GetMode(), and Tcp6GetMode().

#define GET_SND_DATASIZE (Sock) (((Sock)->SndBuffer.DataQueue)->BufSize)

Get the size of the send data.

Parameters:

[in] *Sock* Pointer to the socket.

Returns:

The send data size.

Referenced by TcpDataToSend(), TcpDispatcher(), TcpToSendData(), and TcpTransmitSegment().

#define PROTO_RESERVED_LEN 20

Referenced by SockCreate().

#define PROTO_TOKEN_FORM_SOCK (SockToken, Type) ((Type *) (((SOCK_TOKEN *) (SockToken))->Token))

#define SET_BACKLOG (Sock, Value) ((Sock)->BackLog = (Value))

Set the backlog value of the socket.

Parameters:

[in] *Sock* Pointer to the socket.

[in] *Value* The value to set.

Referenced by TcpConfigurePcb().

#define SET_RCV_BUFFSIZE (Sock, Size) ((Sock)->RcvBuffer.HighWater = (Size))

Set the size of the receive buffer.

Parameters:

[in] *Sock* Pointer to the socket.

[in] *Size* The size to set.

Referenced by TcpConfigurePcb().

#define SET_SND_BUFFSIZE (Sock, Size) ((Sock)->SndBuffer.HighWater = (Size))

Set the size of the send buffer.

Parameters:

[in] *Sock* Pointer to the socket.

[in] *Size* The size to set.

Referenced by TcpConfigurePcb().

#define SO_CLOSED 0
Socket state

Referenced by SockConfigure(), SockConnClosed(), SockDestroy(), SockDestroyChild(), and SockFlush().

#define SO_CONFIGURED_ACTIVE 1

Referenced by TcpConfigurePcb().

#define SO_CONFIGURED_PASSIVE 2

Referenced by TcpConfigurePcb().

#define SO_CONNECTED 3

Referenced by SockConnEstablished().

#define SO_CONNECTING 2

Referenced by SockClone(), SockConnect(), and SockConnEstablished().

#define SO_DISCONNECTING 4

Referenced by SockClose().

#define SO_LISTENING 1

Referenced by TcpConfigurePcb().

#define SO_NO_MAPPING 3

Referenced by Tcp4Configure(), and Tcp6Configure().

#define SO_NO_MORE_DATA 0x0001

#define SO_UNCONFIGURED 0
Socket configure state

Referenced by SockDestroy(), SockDestroyChild(), and SockFlush().

#define SOCK_ABORT 10
Need to reset the protocol process.

Referenced by SockClose(), and TcpDispatcher().

#define SOCK_ATTACH 0
Attach current socket to a new PCB.

The request issued from socket layer to protocol layer.

Referenced by SockCreateChild(), and TcpDispatcher().

#define SOCK_BACKLOG 5

#define SOCK_BUFF_LOW_WATER (2 * 1024)

Referenced by SockCreate().

#define SOCK_CLOSE 9
Need to close the protocol process.

Referenced by SockClose(), and TcpDispatcher().

#define SOCK_CONFIGURE 2
Configure attached PCB.

Referenced by SockConfigure(), and TcpDispatcher().

#define SOCK_CONNECT 8
Need to connect to a peer.

Referenced by SockConnect(), and TcpDispatcher().

#define SOCK_CONSUMED 7
Application has retrieved data from socket.

Referenced by SockRcv(), and TcpDispatcher().

#define SOCK_DETACH 1
Detach current socket from the PCB.

Referenced by SockDestroyChild(), and TcpDispatcher().

#define SOCK_ERROR (Sock, Error) ((Sock)->SockError = (Error))

Set the socket with error state.

Parameters:

[in] *Sock* Pointer to the socket.

[in] *Error* The error state.

Referenced by SockFlush(), SockNoMoreData(), TcpConnectTimeout(), TcpIcmpInput(), TcpInput(), TcpKeepaliveTimeout(), and TcpRexmitTimeout().

#define SOCK_FLUSH 3

Flush attached PCB.

Referenced by SockFlush(), and TcpDispatcher().

#define SOCK_FROM_THIS (a) CR ((a), SOCKET, NetProtocol, SOCK_SIGNATURE)

Referenced by Tcp4Accept(), Tcp4Close(), Tcp4Configure(), Tcp4Connect(), Tcp4GetModeData(), Tcp4Poll(), Tcp4Receive(), Tcp4Routes(), Tcp4Transmit(), Tcp6Accept(), Tcp6Close(), Tcp6Configure(), Tcp6Connect(), Tcp6GetModeData(), Tcp6Poll(), Tcp6Receive(), Tcp6Transmit(), and TcpServiceBindingDestroyChild().

#define SOCK_FROM_TOKEN (Token) (((SOCK_TOKEN *) (Token))->Sock)

#define SOCK_GROUP 14

Need to join a mcast group.

Referenced by SockGroup().

#define SOCK_IS_CLOSED (Sock) ((Sock)->State == SO_CLOSED)

Check whether the socket is closed.

Parameters:

[in] *Sock* Pointer to the socket.

Return values:

TRUE The socket is closed.

FALSE The socket is not closed.

Referenced by SockConnect().

#define SOCK_IS_CONFIGURED (Sock)

Value:

```
(((Sock)->ConfigureState == SO_CONFIGURED_ACTIVE) || \
    ((Sock)->ConfigureState == SO_CONFIGURED_PASSIVE))
```

Check whether the socket is configured.

Parameters:

[in] *Sock* Pointer to the socket

Return values:

TRUE The socket is configued

FALSE The socket is not configued

Referenced by SockConfigure(), SockDestroyChild(), SockFlush(), Tcp4GetMode(), Tcp6GetMode(), and TcpFlushPcb().

#define SOCK_IS_CONFIGURED_ACTIVE (Sock) ((Sock)->ConfigureState == SO_CONFIGURED_ACTIVE)

Check whether the socket is configured to active mode.

Parameters:

[in] *Sock* Pointer to the socket.

Return values:

TRUE The socket is configued to active mode.

FALSE The socket is not configued to active mode.

Referenced by SockConnect().

#define SOCK_IS_CONNECTED (Sock) ((Sock)->State == SO_CONNECTED)

Check whether the socket has connected.

Parameters:

[in] *Sock* Pointer to the socket.

Return values:

TRUE The socket has connected.

FALSE The socket has not connected.

Referenced by SockAccept(), SockRcv(), SockSend(), and SockWakeListenToken().

#define SOCK_IS_CONNECTED_PASSIVE (Sock) ((Sock)->ConfigureState == SO_CONFIGURED_PASSIVE)

Check whether the socket is configured to passive mode.

Parameters:

[in] *Sock* Pointer to the socket.

Return values:

TRUE The socket is configued to passive mode.

FALSE The socket is not configued to passive mode.

#define SOCK_IS_CONNECTING (Sock) ((Sock)->State == SO_CONNECTING)

Check whether the socket is connecting.

Parameters:

[in] *Sock* Pointer to the socket.

Return values:

TRUE The socket is connecting.
FALSE The socket is not connecting.

Referenced by SockRcv(), and SockSend().

#define SOCK_IS_DISCONNECTING (Sock) ((Sock)->State == SO_DISCONNECTING)

Check whether the socket is disconnecting.

Parameters:

[in] *Sock* Pointer to the socket.

Return values:

TRUE The socket is disconnecting.
FALSE The socket is not disconnecting.

Referenced by SockClose().

#define SOCK_IS_LISTENING (Sock) ((Sock)->State == SO_LISTENING)

Check whether the socket is listening.

Parameters:

[in] *Sock* Pointer to the socket.

Return values:

TRUE The socket is listening.
FALSE The socket is not listening.

Referenced by SockAccept(), SockConnFlush(), SockCreate(), and SockWakeListenToken().

#define SOCK_IS_NO_MAPPING (Sock) ((Sock)->ConfigureState == SO_NO_MAPPING)

Check whether the socket is mapped.

Parameters:

[in] *Sock* Pointer to the socket.

Return values:

TRUE The socket is not mapping.
FALSE The socket is mapped.

Referenced by SockAccept(), SockClose(), SockConnect(), SockRcv(), SockRoute(), and SockSend().

#define SOCK_IS_NO_MORE_DATA (Sock) (0 != ((Sock)->Flag & SO_NO_MORE_DATA))

Check whether the socket is no more data.

Parameters:

[in] *Sock* Pointer to the socket.

Return values:

TRUE The socket is no more data.

FALSE The socket still has data.

Referenced by SockRcv().

#define SOCK_IS_UNCONFIGURED (Sock) ((Sock)->ConfigureState == SO_UNCONFIGURED)

Check whether the socket is unconfigured.

Parameters:

[in] *Sock* Pointer to the socket.

Return values:

TRUE The socket is unconfigued.

FALSE The socket is not unconfigued.

Referenced by SockAccept(), SockClose(), SockConnect(), SockDestroy(), SockGroup(), SockRcv(), SockRoute(), and SockSend().

#define SOCK_MODE 13

Need to get the mode data of the protocol.

Referenced by SockGetMode(), and TcpDispatcher().

#define SOCK_NO_MORE_DATA (Sock) ((Sock)->Flag |= SO_NO_MORE_DATA)

Set socket SO_NO_MORE_DATA flag.

Parameters:

[in] *Sock* Pointer to the socket

Referenced by SockNoMoreData().

#define SOCK_POLL 11

Need to poll to the protocol layer.

Referenced by Tcp4Poll(), Tcp6Poll(), and TcpDispatcher().

#define SOCK_RCV_BUF 1

Referenced by SockGetFreeSpace(), and TcpRcvWinNow().

#define SOCK_RCV_BUFF_SIZE (8 * 1024)

#define SOCK_ROUTE 12
Need to add a route information.

Referenced by SockRoute(), and TcpDispatcher().

#define SOCK_SIGNATURE SIGNATURE_32 ('S', 'O', 'C', 'K')

Referenced by SockCreate(), and TcpDestroyChildEntryInHandleBuffer().

#define SOCK_SND 4
Need protocol to send something.

Referenced by SockProcessTcpSndData(), and TcpDispatcher().

#define SOCK_SND_BUF 0

Referenced by SockGetFreeSpace(), SockProcessSndToken(), and SockSend().

#define SOCK_SND_BUFF_SIZE (8 * 1024)

#define SOCK_SNDPUSH 5
Need protocol to send pushed data.

Referenced by SockProcessTcpSndData(), and TcpDispatcher().

#define SOCK_SNDURG 6
Need protocol to send urgent data.

Referenced by SockProcessTcpSndData(), and TcpDispatcher().

Typedef Documentation

typedef union _NET_PROTOCOL NET_PROTOCOL
The union type of TCP and UDP protocol.

typedef struct _SOCK_BUFFER SOCK_BUFFER
The buffer structure of rcvd data and send data used by socket.

typedef struct _SOCK_COMPLETION_TOKEN SOCK_COMPLETION_TOKEN
Socket completion token

typedef EFI_STATUS(* SOCK_CREATE_CALLBACK)(IN SOCKET *This, IN VOID *Context)
The Callback funtion called after the TCP socket is created.

Parameters:

[in]	*This*	Pointer to the socket just created.
[in]	*Context*	Context of the socket.

Return values:

EFI_SUCCESS	This protocol installed successfully.
other	Some error occured.

typedef VOID(* SOCK_DESTROY_CALLBACK)(IN SOCKET *This, IN VOID *Context)
The callback function called before the TCP socket is to be destroyed.

Parameters:

[in]	*This*	The TCP socket to be destroyed.
[in]	*Context*	The context.

typedef struct _SOCK_INIT_DATA SOCK_INIT_DATA
The initialize data for create a new socket.

typedef struct _SOCK_IO_TOKEN SOCK_IO_TOKEN
The application token with data packet

typedef EFI_STATUS(* SOCK_PROTO_HANDLER)(IN SOCKET *Socket, IN UINT8 Request, IN VOID *RequestData)
The handler of protocol for request from socket.

Parameters:

[in]	*Socket*	The socket issuing the request to protocol.
[in]	*Request*	The request issued by socket.
[in]	*RequestData*	The request related data.

Return values:

EFI_SUCCESS	The socket request is completed successfully.
other	The error status returned by the corresponding TCP layer function.

typedef struct _SOCK_TOKEN SOCK_TOKEN
The token structure buffered in socket layer.

typedef struct _TCP_SOCKET SOCKET

typedef struct _TCP_RSV_DATA TCP_RSV_DATA
Reserved data to access the NET_BUF delivered by TCP driver.

Enumeration Type Documentation

enum SOCK_TYPE
The socket type.

Enumerator:

SockDgram This socket providing datagram service.
SockStream This socket providing stream service.

Function Documentation

```
EFI_STATUS SockAccept ( IN SOCKET *   Sock,
                        IN VOID *     Token
                      )
```

Issue a listen token to get an existed connected network instance, or wait for a connection if there is none.

Parameters:

[in] *Sock* Pointer to the socket to accept connections.
[in] *Token* The token to accept a connection.

Return values:

EFI_SUCCESS	Either a connection is accepted or the Token is buffered for further acceptance.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is closed, or the socket is not configured to be a passive one, or the token is already in one of this socket's lists.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.
EFI_OUT_OF_RESOURCE	Failed to buffer the Token due to memory limit.

Issue a listen token to get an existed connected network instance or wait for a connection if there is none.

Parameters:

[in] *Sock* Pointer to the socket to accept connections.
[in] *Token* The token to accept a connection.

Return values:

EFI_SUCCESS	Either a connection is accpeted or the Token is buffered for further acception.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is closed, or the socket is not configured to be a passive one, or the token is already in one of this socket's lists.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.
EFI_OUT_OF_RESOURCE	Failed to buffer the Token due to memory limits.

References _TCP_SOCKET::ConnCnt, _TCP_SOCKET::Parent, SIGNAL_TOKEN, SOCK_IS_CONNECTED, SOCK_IS_LISTENING, SOCK_IS_NO_MAPPING, SOCK_IS_UNCONFIGURED, SockBufferToken(), _TCP_SOCKET::SockHandle, SockStream, and

SockTokenExisted().

Referenced by Tcp4Accept(), and Tcp6Accept().

SOCKET* SockClone (IN SOCKET * *Sock*)

Clone a new socket including its associated protocol control block.

Parameters:

[in] *Sock* Pointer to the socket to be cloned.

Returns:

Pointer to the newly cloned socket. If NULL, an error condition occurred.

Clone a new socket, including its associated protocol control block.

Parameters:

[in] *Sock* Pointer to the socket to be cloned.

Returns:

Pointer to the newly cloned socket. If NULL, an error condition occurred.

References _SOCK_INIT_DATA::BackLog, _TCP_SOCKET::ConfigureState, _SOCK_INIT_DATA::Context, _SOCK_INIT_DATA::CreateCallback, _SOCK_INIT_DATA::DataSize, _SOCK_INIT_DATA::DestroyCallback, _SOCK_INIT_DATA::DriverBinding, _SOCK_INIT_DATA::IpVersion, _SOCK_INIT_DATA::Parent, _SOCK_INIT_DATA::Protocol, _SOCK_INIT_DATA::ProtoData, _SOCK_INIT_DATA::ProtoHandler, _SOCK_INIT_DATA::RcvBufferSize, _SOCK_INIT_DATA::SndBufferSize, SO_CONNECTING, SockCreate(), SockSetState(), _SOCK_INIT_DATA::State, and _SOCK_INIT_DATA::Type.

Referenced by TcpCloneTcb().

```
EFI_STATUS SockClose ( IN OUT SOCKET *   Sock,
                       IN VOID *         Token,
                       IN BOOLEAN        OnAbort
                     )
```

Close or abort the socket associated connection.

Parameters:

[in,out]	*Sock*	Pointer to the socket of the connection to close or abort.
[in]	*Token*	The token for close operation.
[in]	*OnAbort*	TRUE for aborting the connection, FALSE to close it.

Return values:

EFI_SUCCESS	The close or abort operation initialized successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is closed, or the socket is not in a synchronized state , or the token is already in one of this socket's lists.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.

Close or abort the socket associated connection.

Parameters:

[in,out]	*Sock*	Pointer to the socket of the connection to close or abort.
[in]	*Token*	The token for a close operation.
[in]	*OnAbort*	TRUE for aborting the connection; FALSE to close it.

Return values:

EFI_SUCCESS	The close or abort operation initialized successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is closed, or the socket is not in a synchronized state , or the token is already in one of this socket's lists.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.

References SO_DISCONNECTING, SOCK_ABORT, SOCK_CLOSE, SOCK_IS_DISCONNECTING, SOCK_IS_NO_MAPPING, SOCK_IS_UNCONFIGURED, SockSetState(), SockStream, and SockTokenExisted().

Referenced by Tcp4Close(), and Tcp6Close().

```
EFI_STATUS SockConfigure ( IN SOCKET *   Sock,
                           IN VOID *     ConfigData
                         )
```

Configure the specific socket Sock using configuration data ConfigData.

Parameters:

[in]	*Sock*	Pointer to the socket to be configured.
[in]	*ConfigData*	Pointer to the configuration data.

Return values:

EFI_SUCCESS	The socket configured successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is already configured.

References SO_CLOSED, SOCK_CONFIGURE, and SOCK_IS_CONFIGURED.

Referenced by Tcp4Configure(), and Tcp6Configure().

```
VOID SockConnClosed ( IN OUT SOCKET *   Sock   )
```

Called by the low layer protocol to indicate that the connection is closed.

This function flushes the socket, sets the state to SO_CLOSED, and signals the close token.

Parameters:

[in,out]	*Sock*	Pointer to the socket associated with the closed connection.

Called by the low layer protocol to indicate the connection is closed.

This function flushes the socket, sets the state to SO_CLOSED, and signals the close token.

Parameters:

[in,out]	*Sock*	Pointer to the socket associated with the closed connection.

References SIGNAL_TOKEN, SO_CLOSED, SockConnFlush(), SockDestroyChild(), and SockSetState().

Referenced by TcpSetState().

EFI_STATUS SockConnect (IN SOCKET * *Sock*,
IN VOID * *Token*
)

Initiate a connection establishment process.

Parameters:

[in] *Sock* Pointer to the socket to initiate the initate the connection.
[in] *Token* Pointer to the token used for the connection operation.

Return values:

EFI_SUCCESS	The connection initialized successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is closed, or the socket is not configured to be an active one, or the token is already in one of this socket's lists.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.

References SO_CONNECTING, SOCK_CONNECT, SOCK_IS_CLOSED, SOCK_IS_CONFIGURED_ACTIVE, SOCK_IS_NO_MAPPING, SOCK_IS_UNCONFIGURED, SockSetState(), and SockTokenExisted().

Referenced by Tcp4Connect(), and Tcp6Connect().

VOID SockConnEstablished (IN OUT SOCKET * *Sock*)

Called by the low layer protocol to indicate the socket a connection is established.

This function just changes the socket's state to SO_CONNECTED and signals the token used for connection establishment.

Parameters:

[in,out] *Sock* Pointer to the socket associated with the established connection.

References SO_CONNECTED, SO_CONNECTING, SockSetState(), SockWakeConnToken(), and SockWakeListenToken().

Referenced by TcpSetState().

SOCKET* SockCreateChild (IN SOCK_INIT_DATA * *SockInitData*)

Create a socket and its associated protocol control block with the intial data SockInitData and protocol specific data ProtoData.

Parameters:

[in] *SockInitData* Inital data to setting the socket.

Returns:

Pointer to the newly created socket. If NULL, an error condition occured.

References _TCP_SOCKET::Lock, _TCP_SOCKET::ProtoHandler, SOCK_ATTACH, SockCreate(), and SockDestroy().

Referenced by TcpServiceBindingCreateChild().

```
VOID SockDataRcvd ( IN OUT SOCKET *     Sock,
                    IN OUT NET_BUF *    NetBuffer,
                    IN UINT32           UrgLen
                  )
```

Called by the low layer protocol to deliver received data to socket layer.

This function appends the data to the socket receive buffer, set the urgent data length, then checks if any receive token can be signaled.

Parameters:

[in,out]	*Sock*	Pointer to the socket.
[in,out]	*NetBuffer*	Pointer to the buffer that contains the received data.
[in]	*UrgLen*	The length of the urgent data in the received data.

Called by the low layer protocol to deliver received data to socket layer.

This function will append the data to the socket receive buffer, set the urgent data length, and then check if any receive token can be signaled.

Parameters:

[in,out]	*Sock*	Pointer to the socket.
[in,out]	*NetBuffer*	Pointer to the buffer that contains the received data.
[in]	*UrgLen*	The length of the urgent data in the received data.

References SockWakeRcvToken().

Referenced by TcpDeliverData().

```
VOID SockDataSent ( IN OUT SOCKET *  Sock,
                    IN UINT32        Count
                  )
```

Called by low layer protocol to indicate that some data is sent or processed.

This function trims the sent data in the socket send buffer and signals the data token, if proper.

Parameters:

[in,out]	*Sock*	Pointer to the socket.
[in]	*Count*	The length of the data processed or sent, in bytes.

Called by low layer protocol to indicate that some data was sent or processed.

This function trims the sent data in the socket send buffer, and signals the data token if proper.

Parameters:

[in,out] *Sock* Pointer to the socket.
[in] *Count* The length of the data processed or sent, in bytes.

References _SOCK_TOKEN::RemainDataLen, SIGNAL_TOKEN, SockProcessSndToken(), _SOCK_TOKEN::Token, and _SOCK_TOKEN::TokenList.

Referenced by TcpGetSegmentSock().

EFI_STATUS SockDestroyChild (IN OUT SOCKET * *Sock*)
Destroy the socket Sock and its associated protocol control block.

Parameters:
[in,out] *Sock* The socket to be destroyed.

Return values:
EFI_SUCCESS The socket Sock was destroyed successfully.
EFI_ACCESS_DENIED Failed to get the lock to access the socket.

References SO_CLOSED, SO_UNCONFIGURED, SOCK_DETACH, SOCK_IS_CONFIGURED, SockConnFlush(), SockDestroy(), and SockSetState().

Referenced by SockConnClosed(), SockConnFlush(), and TcpServiceBindingDestroyChild().

EFI_STATUS SockFlush (IN OUT SOCKET * *Sock*)
Reset the socket and its associated protocol control block.

Parameters:
[in,out] *Sock* Pointer to the socket to be flushed.

Return values:
EFI_SUCCESS The socket flushed successfully.
EFI_ACCESS_DENIED Failed to get the lock to access the socket.

Reset the socket and its associated protocol control block.

Parameters:
[in,out] *Sock* Pointer to the socket to be flushed.

Return values:
EFI_SUCCESS The socket is flushed successfully.
EFI_ACCESS_DENIED Failed to get the lock to access the socket.

References SO_CLOSED, SO_UNCONFIGURED, SOCK_ERROR, SOCK_FLUSH, SOCK_IS_CONFIGURED, SockConnFlush(), SockSetState(), and SockStream.

Referenced by Tcp4Configure(), and Tcp6Configure().

UINT32 SockGetDataToSend (IN SOCKET * *Sock*,
IN UINT32 *Offset*,
IN UINT32 *Len*,

```
                                OUT UINT8 *   Dest
                              )
```

Called by the low layer protocol to copy some data in socket send buffer starting from the specific offset to a buffer provided by the caller.

Parameters:

[in]	*Sock*	Pointer to the socket.
[in]	*Offset*	The start point of the data to be copied.
[in]	*Len*	The length of the data to be copied.
[out]	*Dest*	Pointer to the destination to copy the data.

Returns:

The data size copied.

Called by the low layer protocol to copy some data in the socket send buffer starting from the specific offset to a buffer provided by the caller.

Parameters:

[in]	*Sock*	Pointer to the socket.
[in]	*Offset*	The start point of the data to be copied.
[in]	*Len*	The length of the data to be copied.
[out]	*Dest*	Pointer to the destination to copy the data.

Returns:

The data size copied.

References SockStream.

Referenced by TcpGetSegmentSock().

```
UINT32 SockGetFreeSpace ( IN SOCKET *   Sock,
                          IN UINT32     Which
                        )
```

Get the length of the free space of the specific socket buffer.

Parameters:

[in]	*Sock*	Pointer to the socket.
[in]	*Which*	Flag to indicate which socket buffer to check: either send buffer or receive buffer.

Returns:

The length of the free space, in bytes.

References _SOCK_BUFFER::DataQueue, _SOCK_BUFFER::HighWater, SOCK_RCV_BUF, and SOCK_SND_BUF.

Referenced by SockProcessSndToken(), SockSend(), and TcpRcvWinNow().

```
EFI_STATUS SockGetMode ( IN SOCKET *      Sock,
                         IN OUT VOID *    Mode
                       )
```

Get the mode data of the low layer protocol.

Parameters:

[in]	*Sock*	Pointer to the socket to get mode data from.
[in,out]	*Mode*	Pointer to the data to store the low layer mode information.

Return values:

EFI_SUCCESS	The mode data was obtained successfully.
EFI_NOT_STARTED	The socket is not configured.

References SOCK_MODE.

Referenced by Tcp4GetModeData(), and Tcp6GetModeData().

```
EFI_STATUS SockGroup ( IN SOCKET *   Sock,
                       IN VOID *     GroupInfo
                     )
```

Configure the low level protocol to join a multicast group for this socket's connection.

Parameters:

[in]	*Sock*	Pointer to the socket of the connection to join the specific multicast group.
[in]	*GroupInfo*	Pointer to the multicast group information.

Return values:

EFI_SUCCESS	The configuration completed successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket.
EFI_NOT_STARTED	The socket is not configured.

Configure the low level protocol to join a multicast group for this socket's connection.

Parameters:

[in]	*Sock*	Pointer to the socket of the connection to join the specific multicast group.
[in]	*GroupInfo*	Pointer to the multicast group info.

Return values:

EFI_SUCCESS	The configuration completed successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket.
EFI_NOT_STARTED	The socket is not configured.

References SOCK_GROUP, and SOCK_IS_UNCONFIGURED.

```
VOID SockNoMoreData ( IN OUT SOCKET *   Sock   )
```

Called by the low layer protocol to indicate that there will be no more data from the communication peer.

This function sets the socket's state to SO_NO_MORE_DATA and signals all queued IO tokens with the error status EFI_CONNECTION_FIN.

Parameters:

[in,out]	*Sock*	Pointer to the socket.

References GET_RCV_DATASIZE, SOCK_ERROR, SOCK_NO_MORE_DATA, and SockFlushPendingToken().

Referenced by TcpDeliverData().

```
EFI_STATUS SockRcv ( IN SOCKET *   Sock,
                     IN VOID *     Token
                   )
```

Issue a token to get data from the socket.

Parameters:

[in]	*Sock*	Pointer to the socket to get data from.
[in]	*Token*	The token to store the received data from the socket.

Return values:

EFI_SUCCESS	The token processed successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is closed, or the socket is not in a synchronized state , or the token is already in one of this socket's lists.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.
EFI_CONNECTION_FIN	The connection is closed and there is no more data.
EFI_OUT_OF_RESOURCE	Failed to buffer the token due to a memory limit.

Issue a token to get data from the socket.

Parameters:

[in]	*Sock*	Pointer to the socket to get data from.
[in]	*Token*	The token to store the received data from the socket.

Return values:

EFI_SUCCESS	The token processed successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is closed, or the socket is not in a synchronized state , or the token is already in one of this socket's lists.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.
EFI_CONNECTION_FIN	The connection is closed and there is no more data.
EFI_OUT_OF_RESOURCE	Failed to buffer the token due to memory limit.

References _SOCK_COMPLETION_TOKEN::Event, GET_RCV_DATASIZE, SIGNAL_TOKEN, SOCK_CONSUMED, SOCK_IS_CONNECTED, SOCK_IS_CONNECTING, SOCK_IS_NO_MAPPING, SOCK_IS_NO_MORE_DATA, SOCK_IS_UNCONFIGURED, SockBufferToken(), SockProcessRcvToken(), SockStream, SockTokenExisted(), and _SOCK_IO_TOKEN::Token.

Referenced by Tcp4Receive(), and Tcp6Receive().

```
EFI_STATUS SockRoute ( IN SOCKET *   Sock,
                       IN VOID *     RouteInfo
                     )
```

Add or remove route information in IP route table associated with this socket.

Parameters:

[in]	*Sock*	Pointer to the socket associated with the IP route table to operate on.
[in]	*RouteInfo*	Pointer to the route information to be processed.

Return values:

EFI_SUCCESS	The route table updated successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.

References SOCK_IS_NO_MAPPING, SOCK_IS_UNCONFIGURED, and SOCK_ROUTE.

Referenced by Tcp4Routes().

```
EFI_STATUS SockSend ( IN SOCKET *   Sock,
                      IN VOID *     Token
                    )
```

Issue a token with data to the socket to send out.

Parameters:

[in]	*Sock*	Pointer to the socket to process the token with data.
[in]	*Token*	The token with data that needs to send out.

Return values:

EFI_SUCCESS	The token processed successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is closed, or the socket is not in a synchronized state , or the token is already in one of this socket's lists.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.
EFI_OUT_OF_RESOURCE	Failed to buffer the token due to a memory limit.

Issue a token with data to the socket to send out.

Parameters:

[in]	*Sock*	Pointer to the socket to process the token with data.
[in]	*Token*	The token with data that needs to send out.

Return values:

EFI_SUCCESS	The token processed successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is closed, or the socket is not in a synchronized state , or the token is already in one of this socket's lists.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.
EFI_OUT_OF_RESOURCE	Failed to buffer the token due to memory limits.

References _SOCK_COMPLETION_TOKEN::Event, _SOCK_IO_TOKEN::Packet, SOCK_IS_CONNECTED, SOCK_IS_CONNECTING, SOCK_IS_NO_MAPPING,

SOCK_IS_UNCONFIGURED, SOCK_SND_BUF, SockBufferToken(), SockGetFreeSpace(), SockProcessTcpSndData(), SockStream, SockTokenExisted(), _SOCK_IO_TOKEN::Token, _SOCK_TOKEN::TokenList, and SOCK_IO_DATA::TxData.

Referenced by Tcp4Transmit(), and Tcp6Transmit().

```
VOID SockSetState ( IN OUT SOCKET *   Sock,
                    IN UINT8          State
                  )
```

Set the state of the socket.

Parameters:

`[in,out]`	*Sock*	Pointer to the socket.
`[in]`	*State*	The new socket state to be set.

Referenced by SockClone(), SockClose(), SockConnClosed(), SockConnect(), SockConnEstablished(), SockDestroy(), SockDestroyChild(), SockFlush(), and TcpConfigurePcb().

NetworkPkg/TcpDxe/SockImpl.c File Reference

Functions

NET_BUF * SockBufFirst (IN SOCK_BUFFER *Sockbuf)
NET_BUF * SockBufNext (IN SOCK_BUFFER *Sockbuf, IN NET_BUF *SockEntry)
VOID EFIAPI SockFreeFoo (IN EFI_EVENT Event)
UINT32 SockTcpDataToRcv (IN SOCK_BUFFER *SockBuffer, OUT BOOLEAN *IsUrg, IN UINT32 BufLen)
VOID SockSetTcpRxData (IN SOCKET *Sock, IN VOID *TcpRxData, IN UINT32 RcvdBytes, IN BOOLEAN IsUrg)
VOID SockProcessSndToken (IN OUT SOCKET *Sock)
UINT32 SockProcessRcvToken (IN OUT SOCKET *Sock, IN OUT SOCK_IO_TOKEN *RcvToken)
EFI_STATUS SockProcessTcpSndData (IN SOCKET *Sock, IN VOID *TcpTxData)
VOID SockFlushPendingToken (IN SOCKET *Sock, IN OUT LIST_ENTRY *PendingTokenList)
VOID SockWakeConnToken (IN OUT SOCKET *Sock)
VOID SockWakeListenToken (IN OUT SOCKET *Sock)
VOID SockWakeRcvToken (IN OUT SOCKET *Sock)
SOCKET * SockCreate (IN SOCK_INIT_DATA *SockInitData)
VOID SockDestroy (IN OUT SOCKET *Sock)
VOID SockConnFlush (IN OUT SOCKET *Sock)
VOID SockSetState (IN OUT SOCKET *Sock, IN UINT8 State)
SOCKET * SockClone (IN SOCKET *Sock)
VOID SockConnEstablished (IN OUT SOCKET *Sock)
VOID SockConnClosed (IN OUT SOCKET *Sock)
VOID SockDataSent (IN OUT SOCKET *Sock, IN UINT32 Count)
UINT32 SockGetDataToSend (IN SOCKET *Sock, IN UINT32 Offset, IN UINT32 Len, OUT UINT8 *Dest)
VOID SockDataRcvd (IN OUT SOCKET *Sock, IN OUT NET_BUF *NetBuffer, IN UINT32 UrgLen)
UINT32 SockGetFreeSpace (IN SOCKET *Sock, IN UINT32 Which)
VOID SockNoMoreData (IN OUT SOCKET *Sock)

Detailed Description

Implementation of the Socket.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

NET_BUF* SockBufFirst (IN SOCK_BUFFER * *Sockbuf*)

Get the first buffer block in the specific socket buffer.

Parameters:

[in] *Sockbuf* Pointer to the socket buffer.

Returns:

Pointer to the first buffer in the queue. NULL if the queue is empty.

Referenced by SockTcpDataToRcv().

NET_BUF* SockBufNext (IN SOCK_BUFFER * *Sockbuf*,
IN NET_BUF * *SockEntry*
)

Get the next buffer block in the specific socket buffer.

Parameters:

[in] *Sockbuf* Pointer to the socket buffer.
[in] *SockEntry* Pointer to the buffer block prior to the required one.

Returns:

Pointer to the buffer block next to SockEntry. NULL if SockEntry is the tail or head entry.

Referenced by SockTcpDataToRcv().

SOCKET* SockClone (IN SOCKET * *Sock*)

Clone a new socket, including its associated protocol control block.

Parameters:

[in] *Sock* Pointer to the socket to be cloned.

Returns:

Pointer to the newly cloned socket. If NULL, an error condition occurred.

References _SOCK_INIT_DATA::BackLog, _TCP_SOCKET::ConfigureState, _SOCK_INIT_DATA::Context, _SOCK_INIT_DATA::CreateCallback, _SOCK_INIT_DATA::DataSize, _SOCK_INIT_DATA::DestroyCallback, _SOCK_INIT_DATA::DriverBinding, _SOCK_INIT_DATA::IpVersion, _SOCK_INIT_DATA::Parent, _SOCK_INIT_DATA::Protocol, _SOCK_INIT_DATA::ProtoData, _SOCK_INIT_DATA::ProtoHandler, _SOCK_INIT_DATA::RcvBufferSize, _SOCK_INIT_DATA::SndBufferSize, SO_CONNECTING, SockCreate(), SockSetState(), _SOCK_INIT_DATA::State, and _SOCK_INIT_DATA::Type.

Referenced by TcpCloneTcb().

VOID SockConnClosed (IN OUT SOCKET * *Sock*)

Called by the low layer protocol to indicate the connection is closed.

This function flushes the socket, sets the state to SO_CLOSED, and signals the close token.

Parameters:

`[in,out]` *Sock* Pointer to the socket associated with the closed connection.

References SIGNAL_TOKEN, SO_CLOSED, SockConnFlush(), SockDestroyChild(), and SockSetState().

Referenced by TcpSetState().

VOID SockConnEstablished (IN OUT SOCKET * *Sock*)

Called by the low layer protocol to indicate the socket a connection is established.

This function just changes the socket's state to SO_CONNECTED and signals the token used for connection establishment.

Parameters:

`[in,out]` *Sock* Pointer to the socket associated with the established connection.

References SO_CONNECTED, SO_CONNECTING, SockSetState(), SockWakeConnToken(), and SockWakeListenToken().

Referenced by TcpSetState().

VOID SockConnFlush (IN OUT SOCKET * *Sock*)

Flush the sndBuffer and rcvBuffer of socket.

Parameters:

`[in,out]` *Sock* Pointer to the socket.

References SIGNAL_TOKEN, SOCK_IS_LISTENING, SockDestroyChild(), and SockFlushPendingToken().

Referenced by SockConnClosed(), SockDestroy(), SockDestroyChild(), and SockFlush().

SOCKET* SockCreate (IN SOCK_INIT_DATA * *SockInitData*)

Create a socket with initial data SockInitData.

Parameters:

`[in]` *SockInitData* Pointer to the initial data of the socket.

Returns:

Pointer to the newly created socket, return NULL when an exception occurs.

References _TCP_SOCKET::BackLog, _TCP_SOCKET::ConnCnt, _TCP_SOCKET::ConnectionList, _TCP_SOCKET::Context, _TCP_SOCKET::CreateCallback, _SOCK_BUFFER::DataQueue, _TCP_SOCKET::DestroyCallback, _TCP_SOCKET::DriverBinding, _SOCK_BUFFER::HighWater, _TCP_SOCKET::IpVersion, _TCP_SOCKET::Link, _TCP_SOCKET::ListenTokenList, _TCP_SOCKET::Lock, _SOCK_BUFFER::LowWater, _TCP_SOCKET::NetProtocol, _TCP_SOCKET::Parent, _TCP_SOCKET::ProcessingSndTokenList, PROTO_RESERVED_LEN, _TCP_SOCKET::ProtoHandler, _TCP_SOCKET::ProtoReserved, _TCP_SOCKET::RcvBuffer, _TCP_SOCKET::RcvTokenList, _TCP_SOCKET::Signature, _TCP_SOCKET::SndBuffer, _TCP_SOCKET::SndTokenList, SOCK_BUFF_LOW_WATER, SOCK_IS_LISTENING,

SOCK_SIGNATURE, _TCP_SOCKET::SockError, _TCP_SOCKET::SockHandle, SockStream, _TCP_SOCKET::State, and _TCP_SOCKET::Type.

Referenced by SockClone(), and SockCreateChild().

```
VOID SockDataRcvd ( IN OUT SOCKET *     Sock,
                    IN OUT NET_BUF *    NetBuffer,
                    IN UINT32           UrgLen
                  )
```

Called by the low layer protocol to deliver received data to socket layer.

This function will append the data to the socket receive buffer, set the urgent data length, and then check if any receive token can be signaled.

Parameters:

[in,out]	*Sock*	Pointer to the socket.
[in,out]	*NetBuffer*	Pointer to the buffer that contains the received data.
[in]	*UrgLen*	The length of the urgent data in the received data.

References SockWakeRcvToken().

Referenced by TcpDeliverData().

```
VOID SockDataSent ( IN OUT SOCKET *  Sock,
                    IN UINT32        Count
                  )
```

Called by low layer protocol to indicate that some data was sent or processed.

This function trims the sent data in the socket send buffer, and signals the data token if proper.

Parameters:

[in,out]	*Sock*	Pointer to the socket.
[in]	*Count*	The length of the data processed or sent, in bytes.

References _SOCK_TOKEN::RemainDataLen, SIGNAL_TOKEN, SockProcessSndToken(), _SOCK_TOKEN::Token, and _SOCK_TOKEN::TokenList.

Referenced by TcpGetSegmentSock().

```
VOID SockDestroy ( IN OUT SOCKET *  Sock )
```

Destroy a socket.

Parameters:

[in,out]	*Sock*	Pointer to the socket.

References SO_CLOSED, SO_UNCONFIGURED, SOCK_IS_UNCONFIGURED, SockConnFlush(),

SockSetState(), and SockStream.

Referenced by SockCreateChild(), and SockDestroyChild().

```
VOID SockFlushPendingToken ( IN SOCKET *            Sock,
                             IN OUT LIST_ENTRY *    PendingTokenList
                           )
```

Flush the tokens in the specific token list.

Parameters:

[in]	*Sock*	Pointer to the socket.
[in,out]	*PendingTokenList*	Pointer to the token list to be flushed.

References SIGNAL_TOKEN, _SOCK_TOKEN::Token, and _SOCK_TOKEN::TokenList.

Referenced by SockConnFlush(), and SockNoMoreData().

```
VOID EFIAPI SockFreeFoo ( IN EFI_EVENT  Event  )
```

User provided callback function for NetbufFromExt.

Parameters:

[in]	*Event*	The Event this notify function registered to, ignored.

Referenced by SockProcessTcpSndData().

```
UINT32 SockGetDataToSend ( IN SOCKET *   Sock,
                           IN UINT32     Offset,
                           IN UINT32     Len,
                           OUT UINT8 *   Dest
                         )
```

Called by the low layer protocol to copy some data in the socket send buffer starting from the specific offset to a buffer provided by the caller.

Parameters:

[in]	*Sock*	Pointer to the socket.
[in]	*Offset*	The start point of the data to be copied.
[in]	*Len*	The length of the data to be copied.
[out]	*Dest*	Pointer to the destination to copy the data.

Returns:

The data size copied.

References SockStream.

Referenced by TcpGetSegmentSock().

```
UINT32 SockGetFreeSpace ( IN SOCKET *   Sock,
                          IN UINT32     Which
```

)

Get the length of the free space of the specific socket buffer.

Parameters:

[in] *Sock* Pointer to the socket.

[in] *Which* Flag to indicate which socket buffer to check: either send buffer or receive buffer.

Returns:

The length of the free space, in bytes.

References _SOCK_BUFFER::DataQueue, _SOCK_BUFFER::HighWater, SOCK_RCV_BUF, and SOCK_SND_BUF.

Referenced by SockProcessSndToken(), SockSend(), and TcpRcvWinNow().

VOID SockNoMoreData (IN OUT SOCKET * *Sock*)

Called by the low layer protocol to indicate that there will be no more data from the communication peer.

This function sets the socket's state to SO_NO_MORE_DATA and signals all queued IO tokens with the error status EFI_CONNECTION_FIN.

Parameters:

[in,out] *Sock* Pointer to the socket.

References GET_RCV_DATASIZE, SOCK_ERROR, SOCK_NO_MORE_DATA, and SockFlushPendingToken().

Referenced by TcpDeliverData().

UINT32 SockProcessRcvToken (IN OUT SOCKET * *Sock*,
IN OUT SOCK_IO_TOKEN * *RcvToken*
)

Get received data from the socket layer to the receive token.

Parameters:

[in,out] *Sock* Pointer to the socket.

[in,out] *RcvToken* Pointer to the application provided receive token.

Returns:

The length of data received in this token.

References SIGNAL_TOKEN, SockSetTcpRxData(), SockStream, and SockTcpDataToRcv().

Referenced by SockRcv(), and SockWakeRcvToken().

VOID SockProcessSndToken (IN OUT SOCKET * *Sock*)

Process the send token.

Parameters:

[in,out] *Sock* Pointer to the socket.

References _SOCK_IO_TOKEN::Packet, SIGNAL_TOKEN, SOCK_SND_BUF, SockGetFreeSpace(), SockProcessTcpSndData(), SockStream, _SOCK_TOKEN::Token, _SOCK_TOKEN::TokenList, and SOCK_IO_DATA::TxData.

Referenced by SockDataSent(), and SockWakeConnToken().

```
EFI_STATUS SockProcessTcpSndData ( IN SOCKET *   Sock,
                                   IN VOID *     TcpTxData
                                 )
```

Process the TCP send data, buffer the tcp txdata, and append the buffer to socket send buffer, then try to send it.

Parameters:

[in]	*Sock*	Pointer to the socket.
[in]	*TcpTxData*	Pointer to the application provided send buffer.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	Failed due to resource limits.

References SOCK_SND, SOCK_SNDPUSH, SOCK_SNDURG, and SockFreeFoo().

Referenced by SockProcessSndToken(), and SockSend().

```
VOID SockSetState ( IN OUT SOCKET *   Sock,
                    IN UINT8          State
                  )
```

Set the state of the socket.

Parameters:

[in,out]	*Sock*	Pointer to the socket.
[in]	*State*	The new socket state to be set.

Referenced by SockClone(), SockClose(), SockConnClosed(), SockConnect(), SockConnEstablished(), SockDestroy(), SockDestroyChild(), SockFlush(), and TcpConfigurePcb().

```
VOID SockSetTcpRxData ( IN SOCKET *   Sock,
                        IN VOID *     TcpRxData,
                        IN UINT32     RcvdBytes,
                        IN BOOLEAN    IsUrg
                      )
```

Copy data from socket buffer to an application provided receive buffer.

Parameters:

[in]	*Sock*	Pointer to the socket.
[in]	*TcpRxData*	Pointer to the application provided receive buffer.
[in]	*RcvdBytes*	The maximum length of the data can be copied.

[in] *IsUrg* If TRUE the data is Out of Bound, FALSE the data is normal.

Referenced by SockProcessRcvToken().

```
UINT32 SockTcpDataToRcv ( IN SOCK_BUFFER * SockBuffer,
                          OUT BOOLEAN * IsUrg,
                          IN UINT32 BufLen )
```

Get the length of the data that can be retrieved from the socket receive buffer.

Parameters:

[in] *SockBuffer* Pointer to the socket receive buffer.

[out] *IsUrg* Pointer to a BOOLEAN variable. If TRUE the data is OOB.

[in] *BufLen* The maximum length of the data buffer to store the received data in the socket layer.

Returns:

The length of the data can be retreived.

References SockBufFirst(), SockBufNext(), and _TCP_RSV_DATA::UrgLen.

Referenced by SockProcessRcvToken().

```
VOID SockWakeConnToken ( IN OUT SOCKET * Sock )
```

Wake up the connection token while the connection is successfully established, then try to process any pending send token.

Parameters:

[in,out] *Sock* Pointer to the socket.

References SIGNAL_TOKEN, and SockProcessSndToken().

Referenced by SockConnEstablished().

```
VOID SockWakeListenToken ( IN OUT SOCKET * Sock )
```

Wake up the listen token while the connection is established successfully.

Parameters:

[in,out] *Sock* Pointer to the socket.

References _TCP_SOCKET::ConnCnt, _TCP_SOCKET::ListenTokenList, _TCP_SOCKET::Parent, SIGNAL_TOKEN, SOCK_IS_CONNECTED, SOCK_IS_LISTENING, _SOCK_TOKEN::Token, and _SOCK_TOKEN::TokenList.

Referenced by SockConnEstablished().

VOID SockWakeRcvToken (IN OUT SOCKET * *Sock*)
Wake up the receive token while some data is received.

Parameters:

`[in,out]` *Sock* Pointer to the socket.

References SockProcessRcvToken(), _SOCK_TOKEN::Token, and _SOCK_TOKEN::TokenList.

Referenced by SockDataRcvd().

Generated on Mon Sep 28 08:49:56 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/TcpDxe/SockImpl.h File Reference

Defines

"memItemRight" valign="bottom">SIGNAL_TOKEN(Token, TokenStatus)

#define SOCK_HEADER_SPACE (60 + 60 + 72)

Functions

EFI_STATUS SockProcessTcpSndData (IN SOCKET *Sock, IN VOID *TcpTxData)

UINT32 SockProcessRcvToken (IN OUT SOCKET *Sock, IN OUT SOCK_IO_TOKEN *RcvToken)

VOID SockConnFlush (IN OUT SOCKET *Sock)

SOCKET * SockCreate (IN SOCK_INIT_DATA *SockInitData)

VOID SockDestroy (IN OUT SOCKET *Sock)

Detailed Description

The function declaration that provided for Socket Interface.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"paramtype">Token,

TokenStatus)

Value:

```
do { \
    (Token)->Status = (TokenStatus); \
    gBS->SignalEvent ((Token)->Event); \
  } while (0)
```

Signal a event with the given status.

Parameters:

[in]	*Token*	The token's event is to be signaled.
[in]	*TokenStatus*	The status to be sent with the event.

Referenced by SockAccept(), SockConnClosed(), SockConnFlush(), SockDataSent(), SockFlushPendingToken(), SockProcessRcvToken(), SockProcessSndToken(), SockRcv(), SockWakeConnToken(), and SockWakeListenToken().

#define SOCK_HEADER_SPACE (60 + 60 + 72)

Function Documentation

VOID SockConnFlush (IN OUT SOCKET * *Sock*)

Flush the sndBuffer and rcvBuffer of socket.

Parameters:

[in,out]	*Sock*	Pointer to the socket.

References SIGNAL_TOKEN, SOCK_IS_LISTENING, SockDestroyChild(), and SockFlushPendingToken().

Referenced by SockConnClosed(), SockDestroy(), SockDestroyChild(), and SockFlush().

SOCKET* SockCreate (IN SOCK_INIT_DATA * *SockInitData*)

Create a socket with initial data SockInitData.

Parameters:

[in]	*SockInitData*	Pointer to the initial data of the socket.

Returns:

Pointer to the newly created socket, return NULL when exception occured.

Create a socket with initial data SockInitData.

Parameters:

[in]	*SockInitData*	Pointer to the initial data of the socket.

Returns:

Pointer to the newly created socket, return NULL when an exception occurs.

References _TCP_SOCKET::BackLog, _TCP_SOCKET::ConnCnt, _TCP_SOCKET::ConnectionList, _TCP_SOCKET::Context, _TCP_SOCKET::CreateCallback, _SOCK_BUFFER::DataQueue, _TCP_SOCKET::DestroyCallback, _TCP_SOCKET::DriverBinding, _SOCK_BUFFER::HighWater, _TCP_SOCKET::IpVersion, _TCP_SOCKET::Link, _TCP_SOCKET::ListenTokenList, _TCP_SOCKET::Lock, _SOCK_BUFFER::LowWater, _TCP_SOCKET::NetProtocol, _TCP_SOCKET::Parent, _TCP_SOCKET::ProcessingSndTokenList, PROTO_RESERVED_LEN, _TCP_SOCKET::ProtoHandler, _TCP_SOCKET::ProtoReserved, _TCP_SOCKET::RcvBuffer, _TCP_SOCKET::RcvTokenList, _TCP_SOCKET::Signature, _TCP_SOCKET::SndBuffer, _TCP_SOCKET::SndTokenList, SOCK_BUFF_LOW_WATER, SOCK_IS_LISTENING,

SOCK_SIGNATURE, _TCP_SOCKET::SockError, _TCP_SOCKET::SockHandle, SockStream, _TCP_SOCKET::State, and _TCP_SOCKET::Type.

Referenced by SockClone(), and SockCreateChild().

```
VOID SockDestroy ( IN OUT SOCKET * Sock )
```

Destroy a socket.

Parameters:

`[in,out]` *Sock* Pointer to the socket.

References SO_CLOSED, SO_UNCONFIGURED, SOCK_IS_UNCONFIGURED, SockConnFlush(), SockSetState(), and SockStream.

Referenced by SockCreateChild(), and SockDestroyChild().

```
UINT32 SockProcessRcvToken ( IN OUT SOCKET *          Sock,
                             IN OUT SOCK_IO_TOKEN *   RcvToken
                           )
```

Get received data from the socket layer to the receive token.

Parameters:

`[in,out]` *Sock* Pointer to the socket.

`[in,out]` *RcvToken* Pointer to the application provided receive token.

Returns:

The length of data received in this token.

References SIGNAL_TOKEN, SockSetTcpRxData(), SockStream, and SockTcpDataToRcv().

Referenced by SockRcv(), and SockWakeRcvToken().

```
EFI_STATUS SockProcessTcpSndData ( IN SOCKET *  Sock,
                                   IN VOID *    TcpTxData
                                 )
```

Process the TCP send data, buffer the tcp txdata and append the buffer to socket send buffer, then try to send it.

Parameters:

`[in]` *Sock* Pointer to the socket.

`[in]` *TcpTxData* Pointer to the application provided send buffer.

Return values:

EFI_SUCCESS The operation completed successfully.

EFI_OUT_OF_RESOURCES Failed due to resource limits.

Process the TCP send data, buffer the tcp txdata, and append the buffer to socket send buffer, then try to send it.

Parameters:

[in]	*Sock*	Pointer to the socket.
[in]	*TcpTxData*	Pointer to the application provided send buffer.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	Failed due to resource limits.

References SOCK_SND, SOCK_SNDPUSH, SOCK_SNDURG, and SockFreeFoo().

Referenced by SockProcessSndToken(), and SockSend().

Generated on Mon Sep 28 08:49:56 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/TcpDxe/SockInterface.c File Reference

Functions

BOOLEAN SockTokenExistedInList (IN LIST_ENTRY *List, IN EFI_EVENT Event)
BOOLEAN SockTokenExisted (IN SOCKET *Sock, IN EFI_EVENT Event)
SOCK_TOKEN * SockBufferToken (IN SOCKET *Sock, IN LIST_ENTRY *List, IN VOID *Token, IN UINT32 DataLen)
EFI_STATUS SockDestroyChild (IN OUT SOCKET *Sock)
SOCKET * SockCreateChild (IN SOCK_INIT_DATA *SockInitData)
EFI_STATUS SockConfigure (IN SOCKET *Sock, IN VOID *ConfigData)
EFI_STATUS SockConnect (IN SOCKET *Sock, IN VOID *Token)
EFI_STATUS SockAccept (IN SOCKET *Sock, IN VOID *Token)
EFI_STATUS SockSend (IN SOCKET *Sock, IN VOID *Token)
EFI_STATUS SockRcv (IN SOCKET *Sock, IN VOID *Token)
EFI_STATUS SockFlush (IN OUT SOCKET *Sock)
EFI_STATUS SockClose (IN OUT SOCKET *Sock, IN VOID *Token, IN BOOLEAN OnAbort)
EFI_STATUS SockGetMode (IN SOCKET *Sock, IN OUT VOID *Mode)
EFI_STATUS SockGroup (IN SOCKET *Sock, IN VOID *GroupInfo)
EFI_STATUS SockRoute (IN SOCKET *Sock, IN VOID *RouteInfo)

Detailed Description

Interface function of the Socket.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS SockAccept ( IN SOCKET *   Sock,
                        IN VOID *     Token
                      )
```

Issue a listen token to get an existed connected network instance or wait for a connection if there is none.

Parameters:

[in] *Sock* Pointer to the socket to accept connections.
[in] *Token* The token to accept a connection.

Return values:

EFI_SUCCESS	Either a connection is accpeted or the Token is buffered for further acception.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is closed, or the socket is not configured to be a passive one, or the token is already in one of this socket's lists.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.
EFI_OUT_OF_RESOURCE	Failed to buffer the Token due to memory limits.

References _TCP_SOCKET::ConnCnt, _TCP_SOCKET::Parent, SIGNAL_TOKEN, SOCK_IS_CONNECTED, SOCK_IS_LISTENING, SOCK_IS_NO_MAPPING, SOCK_IS_UNCONFIGURED, SockBufferToken(), _TCP_SOCKET::SockHandle, SockStream, and SockTokenExisted().

Referenced by Tcp4Accept(), and Tcp6Accept().

```
SOCK_TOKEN* SockBufferToken ( IN SOCKET *          Sock,
                              IN LIST_ENTRY *      List,
                              IN VOID *            Token,
                              IN UINT32            DataLen
                            )
```

Buffer a token into the specific list of the socket Sock.

Parameters:

[in]	*Sock*	Pointer to the instance's socket.
[in]	*List*	Pointer to the list to store the token.
[in]	*Token*	Pointer to the token to be buffered.
[in]	*DataLen*	The data length of the buffer contained in Token.

Returns:

Pointer to the token that wraps Token. If NULL, an error condition occurred.

References _SOCK_TOKEN::RemainDataLen, _SOCK_TOKEN::Sock, _SOCK_TOKEN::Token, and _SOCK_TOKEN::TokenList.

Referenced by SockAccept(), SockRcv(), and SockSend().

```
EFI_STATUS SockClose ( IN OUT SOCKET *   Sock,
                       IN VOID *         Token,
                       IN BOOLEAN        OnAbort
                     )
```

Close or abort the socket associated connection.

Parameters:

[in,out]	*Sock*	Pointer to the socket of the connection to close or abort.
[in]	*Token*	The token for a close operation.
[in]	*OnAbort*	TRUE for aborting the connection; FALSE to close it.

Return values:

EFI_SUCCESS	The close or abort operation initialized successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is closed, or the socket is not in a synchronized state , or the token is already in one of this socket's lists.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.

References SO_DISCONNECTING, SOCK_ABORT, SOCK_CLOSE, SOCK_IS_DISCONNECTING, SOCK_IS_NO_MAPPING, SOCK_IS_UNCONFIGURED, SockSetState(), SockStream, and SockTokenExisted().

Referenced by Tcp4Close(), and Tcp6Close().

```
EFI_STATUS SockConfigure ( IN SOCKET *   Sock,
                           IN VOID *     ConfigData
                         )
```

Configure the specific socket Sock using configuration data ConfigData.

Parameters:

[in]	*Sock*	Pointer to the socket to be configured.
[in]	*ConfigData*	Pointer to the configuration data.

Return values:

EFI_SUCCESS	The socket configured successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is already configured.

References SO_CLOSED, SOCK_CONFIGURE, and SOCK_IS_CONFIGURED.

Referenced by Tcp4Configure(), and Tcp6Configure().

```
EFI_STATUS SockConnect ( IN SOCKET *   Sock,
                         IN VOID *     Token
                       )
```

Initiate a connection establishment process.

Parameters:

[in]	*Sock*	Pointer to the socket to initiate the initate the connection.
[in]	*Token*	Pointer to the token used for the connection operation.

Return values:

EFI_SUCCESS	The connection initialized successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is closed, or the socket is not configured to be an active one, or the token is already in one of this socket's lists.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.

References SO_CONNECTING, SOCK_CONNECT, SOCK_IS_CLOSED, SOCK_IS_CONFIGURED_ACTIVE, SOCK_IS_NO_MAPPING, SOCK_IS_UNCONFIGURED, SockSetState(), and SockTokenExisted().

Referenced by Tcp4Connect(), and Tcp6Connect().

SOCKET* SockCreateChild (IN SOCK_INIT_DATA * *SockInitData*)

Create a socket and its associated protocol control block with the intial data SockInitData and protocol specific data ProtoData.

Parameters:

`[in]` *SockInitData* Inital data to setting the socket.

Returns:

Pointer to the newly created socket. If NULL, an error condition occured.

References _TCP_SOCKET::Lock, _TCP_SOCKET::ProtoHandler, SOCK_ATTACH, SockCreate(), and SockDestroy().

Referenced by TcpServiceBindingCreateChild().

EFI_STATUS SockDestroyChild (IN OUT SOCKET * *Sock*)

Destroy the socket Sock and its associated protocol control block.

Parameters:

`[in,out]` *Sock* The socket to be destroyed.

Return values:

EFI_SUCCESS	The socket Sock was destroyed successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket.

References SO_CLOSED, SO_UNCONFIGURED, SOCK_DETACH, SOCK_IS_CONFIGURED, SockConnFlush(), SockDestroy(), and SockSetState().

Referenced by SockConnClosed(), SockConnFlush(), and TcpServiceBindingDestroyChild().

EFI_STATUS SockFlush (IN OUT SOCKET * *Sock*)

Reset the socket and its associated protocol control block.

Parameters:

`[in,out]` *Sock* Pointer to the socket to be flushed.

Return values:

EFI_SUCCESS	The socket is flushed successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket.

References SO_CLOSED, SO_UNCONFIGURED, SOCK_ERROR, SOCK_FLUSH, SOCK_IS_CONFIGURED, SockConnFlush(), SockSetState(), and SockStream.

Referenced by Tcp4Configure(), and Tcp6Configure().

```
EFI_STATUS SockGetMode ( IN SOCKET *      Sock,
                         IN OUT VOID *    Mode
                       )
```

Get the mode data of the low layer protocol.

Parameters:

[in]	*Sock*	Pointer to the socket to get mode data from.
[in,out]	*Mode*	Pointer to the data to store the low layer mode information.

Return values:

EFI_SUCCESS	The mode data was obtained successfully.
EFI_NOT_STARTED	The socket is not configured.

References SOCK_MODE.

Referenced by Tcp4GetModeData(), and Tcp6GetModeData().

```
EFI_STATUS SockGroup ( IN SOCKET *   Sock,
                       IN VOID *     GroupInfo
                     )
```

Configure the low level protocol to join a multicast group for this socket's connection.

Parameters:

[in]	*Sock*	Pointer to the socket of the connection to join the specific multicast group.
[in]	*GroupInfo*	Pointer to the multicast group info.

Return values:

EFI_SUCCESS	The configuration completed successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket.
EFI_NOT_STARTED	The socket is not configured.

References SOCK_GROUP, and SOCK_IS_UNCONFIGURED.

```
EFI_STATUS SockRcv ( IN SOCKET *   Sock,
                     IN VOID *     Token
                   )
```

Issue a token to get data from the socket.

Parameters:

[in]	*Sock*	Pointer to the socket to get data from.
[in]	*Token*	The token to store the received data from the socket.

Return values:

EFI_SUCCESS	The token processed successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is closed, or the socket is not in a synchronized state , or the token is already in one of this socket's lists.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.

EFI_CONNECTION_FIN	The connection is closed and there is no more data.
EFI_OUT_OF_RESOURCE	Failed to buffer the token due to memory limit.

References _SOCK_COMPLETION_TOKEN::Event, GET_RCV_DATASIZE, SIGNAL_TOKEN, SOCK_CONSUMED, SOCK_IS_CONNECTED, SOCK_IS_CONNECTING, SOCK_IS_NO_MAPPING, SOCK_IS_NO_MORE_DATA, SOCK_IS_UNCONFIGURED, SockBufferToken(), SockProcessRcvToken(), SockStream, SockTokenExisted(), and _SOCK_IO_TOKEN::Token.

Referenced by Tcp4Receive(), and Tcp6Receive().

```
EFI_STATUS SockRoute ( IN SOCKET *   Sock,
                       IN VOID *     RouteInfo
                     )
```

Add or remove route information in IP route table associated with this socket.

Parameters:

[in]	*Sock*	Pointer to the socket associated with the IP route table to operate on.
[in]	*RouteInfo*	Pointer to the route information to be processed.

Return values:

EFI_SUCCESS	The route table updated successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.

References SOCK_IS_NO_MAPPING, SOCK_IS_UNCONFIGURED, and SOCK_ROUTE.

Referenced by Tcp4Routes().

```
EFI_STATUS SockSend ( IN SOCKET *   Sock,
                      IN VOID *     Token
                    )
```

Issue a token with data to the socket to send out.

Parameters:

[in]	*Sock*	Pointer to the socket to process the token with data.
[in]	*Token*	The token with data that needs to send out.

Return values:

EFI_SUCCESS	The token processed successfully.
EFI_ACCESS_DENIED	Failed to get the lock to access the socket, or the socket is closed, or the socket is not in a synchronized state , or the token is already in one of this socket's lists.
EFI_NO_MAPPING	The IP address configuration operation is not finished.
EFI_NOT_STARTED	The socket is not configured.
EFI_OUT_OF_RESOURCE	Failed to buffer the token due to memory limits.

References _SOCK_COMPLETION_TOKEN::Event, _SOCK_IO_TOKEN::Packet, SOCK_IS_CONNECTED, SOCK_IS_CONNECTING, SOCK_IS_NO_MAPPING, SOCK_IS_UNCONFIGURED, SOCK_SND_BUF, SockBufferToken(), SockGetFreeSpace(),

SockProcessTcpSndData(), SockStream, SockTokenExisted(), _SOCK_IO_TOKEN::Token, _SOCK_TOKEN::TokenList, and SOCK_IO_DATA::TxData.

Referenced by Tcp4Transmit(), and Tcp6Transmit().

```
BOOLEAN SockTokenExisted ( IN SOCKET *     Sock,
                           IN EFI_EVENT    Event
                         )
```

Call SockTokenExistedInList() to check whether the Event is in the related socket's lists.

Parameters:

[in] *Sock* Pointer to the instance's socket.
[in] *Event* The event to be checked.

Return values:

TRUE The Event exists in related socket's lists.
FALSE The Event is not in related socket's lists.

References SockTokenExistedInList().

Referenced by SockAccept(), SockClose(), SockConnect(), SockRcv(), and SockSend().

```
BOOLEAN SockTokenExistedInList ( IN LIST_ENTRY *  List,
                                 IN EFI_EVENT     Event
                               )
```

Check whether the Event is in the List.

Parameters:

[in] *List* Pointer to the token list to be searched.
[in] *Event* The event to be checked.

Return values:

TRUE The specific Event exists in the List.
FALSE The specific Event is not in the List.

References _SOCK_COMPLETION_TOKEN::Event, and _SOCK_TOKEN::Token.

Referenced by SockTokenExisted().

NetworkPkg/TcpDxe/TcpDispatcher.c File Reference

Functions

EFI_STATUS Tcp4Route (IN TCP_CB *Tcb, IN TCP4_ROUTE_INFO *RouteInfo)
EFI_STATUS Tcp4GetMode (IN TCP_CB *Tcb, IN OUT TCP4_MODE_DATA *Mode)
EFI_STATUS Tcp6GetMode (IN TCP_CB *Tcb, IN OUT TCP6_MODE_DATA *Mode)
EFI_STATUS TcpBind (IN TCP_ACCESS_POINT *TcpAp, IN UINT8 IpVersion)
VOID TcpFlushPcb (IN OUT TCP_CB *Tcb)
EFI_STATUS TcpAttachPcb (IN SOCKET *Sk)
VOID TcpDetachPcb (IN OUT SOCKET *Sk)
EFI_STATUS TcpConfigurePcb (IN SOCKET *Sk, IN TCP_CONFIG_DATA *CfgData)
EFI_STATUS TcpDispatcher (IN SOCKET *Sock, IN UINT8 Request, IN VOID *Data)

Detailed Description

The implementation of a dispatch routine for processing TCP requests.

(C) Copyright 2014 Hewlett-Packard Development Company, L.P.
Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS Tcp4GetMode ( IN TCP_CB *              Tcb,
                         IN OUT TCP4_MODE_DATA *  Mode
                       )
```

Get the operational settings of this TCPv4 instance.

Parameters:

[in]	*Tcb*	Pointer to the TCP_CB of this TCP instance.
[in,out]	*Mode*	Pointer to the buffer to store the operational settings.

Return values:

EFI_SUCCESS	The mode data was read.
EFI_NOT_STARTED	No configuration data is available because this instance hasn't been started.

References GET_BACKLOG, GET_RCV_BUFFSIZE, GET_SND_BUFFSIZE, SOCK_IS_CONFIGURED, _TCP_SOCKET::State, TCP_CTRL_NO_NAGLE, TCP_CTRL_NO_TS, TCP_CTRL_NO_WS, TCP_FLG_ON, TCP_LISTEN, and TCP_TICK_HZ.

Referenced by TcpDispatcher().

```
EFI_STATUS Tcp4Route ( IN TCP_CB *              Tcb,
                       IN TCP4_ROUTE_INFO *     RouteInfo
                     )
```

Add or remove a route entry in the IP route table associated with this TCP instance.

Parameters:

[in]	*Tcb*	Pointer to the TCP_CB of this TCP instance.
[in]	*RouteInfo*	Pointer to the route information to be processed.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_STARTED	The driver instance has not been started.
EFI_NO_MAPPING	When using the default address, configuration(DHCP, BOOTP, RARP, etc.) is not finished yet.
EFI_OUT_OF_RESOURCES	Could not add the entry to the routing table.
EFI_NOT_FOUND	This route is not in the routing table (when RouteInfo->DeleteRoute is TRUE).
EFI_ACCESS_DENIED	The route is already defined in the routing table (when RouteInfo->DeleteRoute is FALSE).

Referenced by TcpDispatcher().

```
EFI_STATUS Tcp6GetMode ( IN TCP_CB *              Tcb,
                         IN OUT TCP6_MODE_DATA *  Mode
                       )
```

Get the operational settings of this TCPv6 instance.

Parameters:

[in]	*Tcb*	Pointer to the TCP_CB of this TCP instance.
[in,out]	*Mode*	Pointer to the buffer to store the operational settings.

Return values:

EFI_SUCCESS	The mode data was read.
EFI_NOT_STARTED	No configuration data is available because this instance hasn't been started.

References GET_BACKLOG, GET_RCV_BUFFSIZE, GET_SND_BUFFSIZE, SOCK_IS_CONFIGURED, TCP_CTRL_NO_NAGLE, TCP_CTRL_NO_TS, TCP_CTRL_NO_WS, TCP_FLG_ON, TCP_LISTEN, and TCP_TICK_HZ.

Referenced by TcpDispatcher().

```
EFI_STATUS TcpAttachPcb ( IN SOCKET *  Sk  )
```

Attach a Pcb to the socket.

Parameters:

[in]	*Sk*	Pointer to the socket of this TCP instance.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	Failed due to resource limits.

References _TCP_CONTROL_BLOCK::IpInfo, _TCP_SERVICE_DATA::IpIo, _TCP_CONTROL_BLOCK::List, _TCP_CONTROL_BLOCK::RcvQue, _TCP_CONTROL_BLOCK::Sk, _TCP_CONTROL_BLOCK::SndQue, _TCP_CONTROL_BLOCK::State, TCP_CLOSED, _TCP_PROTO_DATA::TcpPcb, and _TCP_PROTO_DATA::TcpService.

Referenced by TcpDispatcher().

```
EFI_STATUS TcpBind ( IN TCP_ACCESS_POINT *   TcpAp,
                     IN UINT8                IpVersion
                   )
```

If TcpAp->StationPort isn't zero, check whether the access point is registered, else generate a random station port for this access point.

Parameters:

[in]	*TcpAp*	Pointer to the access point.
[in]	*IpVersion*	IP_VERSION_4 or IP_VERSION_6

Return values:

EFI_SUCCESS	The check passed or the port is assigned.
EFI_INVALID_PARAMETER	The non-zero station port is already used.
EFI_OUT_OF_RESOURCES	No port can be allocated.

References mTcp4RandomPort, mTcp6RandomPort, TCP_PORT_KNOWN, TCP_PORT_USER_RESERVED, and TcpFindTcbByPeer().

Referenced by TcpConfigurePcb().

```
EFI_STATUS TcpConfigurePcb ( IN SOCKET *            Sk,
                             IN TCP_CONFIG_DATA *   CfgData
                           )
```

Configure the Pcb using CfgData.

Parameters:

[in]	*Sk*	Pointer to the socket of this TCP instance.
[in]	*CfgData*	Pointer to the TCP configuration data.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_INVALID_PARAMETER	A same access point has been configured in another TCP instance.
EFI_OUT_OF_RESOURCES	Failed due to resource limits.

References _TCP_CONTROL_BLOCK::CongestState, _TCP_CONTROL_BLOCK::ConnectTimeout, _TCP_CONTROL_BLOCK::CtrlFlag, _TCP_CONTROL_BLOCK::CWnd, _TCP_CONTROL_BLOCK::FinWait2Timeout, _TCP_PEER::Ip, _TCP_CONTROL_BLOCK::IpInfo, _TCP_CONTROL_BLOCK::KeepAliveIdle, _TCP_CONTROL_BLOCK::KeepAlivePeriod, _TCP_CONTROL_BLOCK::LocalEnd, _TCP_CONTROL_BLOCK::MaxKeepAlive,

_TCP_CONTROL_BLOCK::MaxRexmit, _TCP_PEER::Port, _TCP_CONTROL_BLOCK::RcvMss, _TCP_CONTROL_BLOCK::RcvQue, _TCP_CONTROL_BLOCK::RemoteEnd, _TCP_CONTROL_BLOCK::RemoteIpZero, _TCP_CONTROL_BLOCK::Rto, SET_BACKLOG, SET_RCV_BUFFSIZE, SET_SND_BUFFSIZE, _TCP_CONTROL_BLOCK::SndMss, _TCP_CONTROL_BLOCK::SndQue, SO_CONFIGURED_ACTIVE, SO_CONFIGURED_PASSIVE, SO_LISTENING, SockSetState(), _TCP_CONTROL_BLOCK::SRtt, _TCP_CONTROL_BLOCK::Ssthresh, _TCP_CONTROL_BLOCK::State, _TCP_CONTROL_BLOCK::SubnetMask, TCP6_REFRESH_NEIGHBOR_TICK, TCP_BACKLOG, TCP_BACKLOG_MIN, TCP_CLEAR_FLG, TCP_CLOSED, TCP_COMP_VAL, TCP_CONGEST_OPEN, TCP_CONNECT_TIME, TCP_CONNECT_TIME_MIN, TCP_CTRL_NO_KEEPALIVE, TCP_CTRL_NO_NAGLE, TCP_CTRL_NO_TS, TCP_CTRL_NO_WS, TCP_FIN_WAIT2_TIME, TCP_FIN_WAIT2_TIME_MAX, TCP_KEEPALIVE_IDLE_MAX, TCP_KEEPALIVE_IDLE_MIN, TCP_KEEPALIVE_PERIOD, TCP_KEEPALIVE_PERIOD_MIN, TCP_LISTEN, TCP_MAX_KEEPALIVE, TCP_MAX_KEEPALIVE_MIN, TCP_MAX_LOSS, TCP_MAX_LOSS_MIN, TCP_RCV_BUF_SIZE, TCP_RCV_BUF_SIZE_MIN, TCP_SET_FLG, TCP_SND_BUF_SIZE, TCP_SND_BUF_SIZE_MIN, TCP_TICK_HZ, TCP_TIME_WAIT_TIME, TCP_TIME_WAIT_TIME_MAX, TcpBind(), TcpGetRcvMss(), TcpInsertTcb(), TcpInstallDevicePath(), _TCP_PROTO_DATA::TcpPcb, TcpSetState(), _TCP_CONTROL_BLOCK::Tick, _TCP_CONTROL_BLOCK::TimeWaitTimeout, _TCP_CONTROL_BLOCK::Tos, _TCP_CONTROL_BLOCK::Ttl, and _TCP_CONTROL_BLOCK::UseDefaultAddr.

Referenced by TcpDispatcher().

VOID TcpDetachPcb (IN OUT SOCKET * *Sk*)

Detach the Pcb of the socket.

Parameters:

`[in,out]` *Sk* Pointer to the socket of this TCP instance.

References _TCP_CONTROL_BLOCK::IpInfo, _TCP_SERVICE_DATA::IpIo, TcpFlushPcb(), _TCP_PROTO_DATA::TcpPcb, and _TCP_PROTO_DATA::TcpService.

Referenced by TcpDispatcher().

EFI_STATUS TcpDispatcher (IN SOCKET * *Sock*,
IN UINT8 *Request*,
IN VOID * *Data*
)

The procotol handler provided to the socket layer, which is used to dispatch the socket level requests by calling the corresponding TCP layer functions.

Parameters:

`[in]` *Sock* Pointer to the socket of this TCP instance.

`[in]` *Request* The code of this operation request.

`[in]` *Data* Pointer to the operation specific data passed in together with the operation request. This is an optional parameter that may be NULL.

Return values:

EFI_SUCCESS The socket request completed successfully.

other The error status returned by the corresponding TCP layer function.

References _TCP_CONTROL_BLOCK::CtrlFlag, GET_SND_DATASIZE, _TCP_SERVICE_DATA::IpIo, _TCP_SOCKET::IpVersion, _TCP_CONTROL_BLOCK::Sk, _TCP_CONTROL_BLOCK::SndPsh, _TCP_CONTROL_BLOCK::SndUp, SOCK_ABORT, SOCK_ATTACH, SOCK_CLOSE, SOCK_CONFIGURE, SOCK_CONNECT, SOCK_CONSUMED, SOCK_DETACH, SOCK_FLUSH, SOCK_MODE, SOCK_POLL, SOCK_ROUTE, SOCK_SND, SOCK_SNDPUSH, SOCK_SNDURG, Tcp4GetMode(), Tcp4Route(), Tcp6GetMode(), TCP_CTRL_SND_PSH, TCP_CTRL_SND_URG, TCP_SET_FLG, TcpAttachPcb(), TcpConfigurePcb(), TcpDetachPcb(), TcpFlushPcb(), TcpGetMaxSndNxt(), TcpOnAppAbort(), TcpOnAppClose(), TcpOnAppConnect(), TcpOnAppConsume(), TcpOnAppSend(), _TCP_PROTO_DATA::TcpPcb, and _TCP_PROTO_DATA::TcpService.

VOID TcpFlushPcb (IN OUT TCP_CB * *Tcb*)

Flush the Tcb add its associated protocols.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB to be flushed.

References _TCP_SOCKET::DevicePath, SOCK_IS_CONFIGURED, _TCP_SOCKET::SockHandle, and TCP_CLOSED.

Referenced by TcpDetachPcb(), and TcpDispatcher().

Generated on Mon Sep 28 08:49:56 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/TcpDxe/TcpDriver.c File Reference

Functions

EFI_STATUS TcpCreateTimer (VOID)

VOID TcpDestroyTimer (VOID)

EFI_STATUS EFIAPI TcpDriverEntryPoint (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)

EFI_STATUS TcpCreateService (IN EFI_HANDLE Controller, IN EFI_HANDLE Image, IN UINT8 IpVersion)

EFI_STATUS EFIAPI TcpDestroyChildEntryInHandleBuffer (IN LIST_ENTRY *Entry, IN VOID *Context)

EFI_STATUS TcpDestroyService (IN EFI_HANDLE Controller, IN EFI_HANDLE ImageHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer, IN UINT8 IpVersion)

EFI_STATUS EFIAPI Tcp4DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI Tcp4DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI Tcp4DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)

EFI_STATUS EFIAPI Tcp6DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI Tcp6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI Tcp6DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)

EFI_STATUS TcpCreateSocketCallback (IN SOCKET *This, IN VOID *Context)

VOID TcpDestroySocketCallback (IN SOCKET *This, IN VOID *Context)

EFI_STATUS EFIAPI TcpServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN OUT EFI_HANDLE *ChildHandle)

EFI_STATUS EFIAPI TcpServiceBindingDestroyChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE ChildHandle)

Variables

UINT16	mTcp4RandomPort
UINT16	mTcp6RandomPort
TCP_HEARTBEAT_TIMER	mTcpTimer
EFI_TCP4_PROTOCOL	gTcp4ProtocolTemplate
EFI_TCP6_PROTOCOL	gTcp6ProtocolTemplate
SOCK_INIT_DATA	mTcpDefaultSockData
EFI_DRIVER_BINDING_PROTOCOL	gTcp4DriverBinding
EFI_DRIVER_BINDING_PROTOCOL	gTcp6DriverBinding
EFI_SERVICE_BINDING_PROTOCOL	gTcpServiceBinding

Detailed Description

The driver binding and service binding protocol for the TCP driver.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI
Tcp4DriverBindingStart (
  IN EFI_DRIVER_BINDING_PROTOCOL *   This,
  IN EFI_HANDLE                      ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL *      RemainingDevicePath
)
```

Start this driver on ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to bind driver to.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	The driver is added to ControllerHandle.
EFI_OUT_OF_RESOURCES	There are not enough resources to start the driver.
other	The driver cannot be added to ControllerHandle.

References TcpCreateService().

```
EFI_STATUS EFIAPI
Tcp4DriverBindingStop (
  IN EFI_DRIVER_BINDING_PROTOCOL *   This,
  IN EFI_HANDLE                      ControllerHandle,
```

```
                                IN UINTN                          NumberOfChildren,
                                IN EFI_HANDLE *                   ChildHandleBuffer
                              )
```

Stop this driver on ControllerHandle.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
[in]	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
[in]	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References TcpDestroyService().

```
EFI_STATUS EFIAPI
Tcp4DriverBindingSupported    ( IN
                                EFI_DRIVER_BINDING_PROTOCOL *     This,
                                IN EFI_HANDLE                     ControllerHandle,
                                IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
                              )
```

Test to see if this driver supports ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to test.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	This driver supports this device.
EFI_ALREADY_STARTED	This driver is already running on this device.
other	This driver does not support this device.

```
EFI_STATUS EFIAPI
Tcp6DriverBindingStart        ( IN EFI_DRIVER_BINDING_PROTOCOL
                                *                                 This,
                                IN EFI_HANDLE                     ControllerHandle,
                                IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
                              )
```

Start this driver on ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to bind driver to.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	The driver is added to ControllerHandle.
EFI_OUT_OF_RESOURCES	There are not enough resources to start the driver.
other	The driver cannot be added to ControllerHandle.

References TcpCreateService().

```
EFI_STATUS EFIAPI
Tcp6DriverBindingStop ( IN EFI_DRIVER_BINDING_PROTOCOL *  This,
                        IN EFI_HANDLE                     ControllerHandle,
                        IN UINTN                          NumberOfChildren,
                        IN EFI_HANDLE *                   ChildHandleBuffer
                      )
```

Stop this driver on ControllerHandle.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
[in]	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
[in]	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References TcpDestroyService().

```
EFI_STATUS EFIAPI
Tcp6DriverBindingSupported ( IN EFI_DRIVER_BINDING_PROTOCOL *  This,
                             IN EFI_HANDLE                     ControllerHandle,
                             IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
                           )
```

Test to see if this driver supports ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to test.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	This driver supports this device.
EFI_ALREADY_STARTED	This driver is already running on this device.
other	This driver does not support this device.

```
EFI_STATUS TcpCreateService ( IN EFI_HANDLE  Controller,
                              IN EFI_HANDLE  Image,
                              IN UINT8       IpVersion
```

)

Create a new TCP4 or TCP6 driver service binding protocol

Parameters:

[in]	*Controller*	Controller handle of device to bind driver to.
[in]	*Image*	The TCP driver's image handle.
[in]	*IpVersion*	IP_VERSION_4 or IP_VERSION_6.

Return values:

EFI_OUT_OF_RESOURCES	Failed to allocate some resources.
EFI_SUCCESS	A new IP6 service binding private was created.

References _TCP_SERVICE_DATA::ControllerHandle, _TCP_SERVICE_DATA::DriverBindingHandle, gTcpServiceBinding, _TCP_SERVICE_DATA::IpIo, _TCP_SERVICE_DATA::IpVersion, _TCP_SERVICE_DATA::ServiceBinding, _TCP_SERVICE_DATA::Signature, _TCP_SERVICE_DATA::SocketList, TCP_DRIVER_SIGNATURE, TcpCreateTimer(), TcpDestroyTimer(), and TcpRxCallback().

Referenced by Tcp4DriverBindingStart(), and Tcp6DriverBindingStart().

EFI_STATUS TcpCreateSocketCallback (IN SOCKET * *This*,
IN VOID * *Context*
)

The Callback funtion called after the TCP socket was created.

Parameters:

[in]	*This*	Pointer to the socket just created
[in]	*Context*	Context of the socket

Return values:

EFI_SUCCESS	This protocol installed successfully.
other	An error occured.

References _TCP_SERVICE_DATA::ControllerHandle, _TCP_SERVICE_DATA::DriverBindingHandle, _TCP_SERVICE_DATA::IpIo, and _TCP_SERVICE_DATA::SocketList.

EFI_STATUS TcpCreateTimer (VOID)

Create and start the heartbeat timer for the TCP driver.

Return values:

EFI_SUCCESS	The timer was successfully created and started.
other	The timer was not created.

References _TCP_HEARTBEAT_TIMER::RefCnt, TCP_TICK_HZ, TcpTicking(), and _TCP_HEARTBEAT_TIMER::TimerEvent.

Referenced by TcpCreateService().

EFI_STATUS EFIAPI TcpDestroyChildEntryInHandleBuffer (IN LIST_ENTRY * *Entry*,

```
                                    IN VOID *          Context
                                  )
```

Callback function which provided by user to remove one node in NetDestroyLinkList process.

Parameters:

[in]	*Entry*	The entry to be removed.
[in]	*Context*	Pointer to the callback context corresponds to the Context in NetDestroyLinkList.

Return values:

EFI_SUCCESS	The entry has been removed successfully.
Others	Fail to remove the entry.

References SOCK_SIGNATURE, and _TCP_SOCKET::SockHandle.

Referenced by TcpDestroyService().

```
EFI_STATUS TcpDestroyService ( IN EFI_HANDLE      Controller,
                               IN EFI_HANDLE      ImageHandle,
                               IN UINTN           NumberOfChildren,
                               IN EFI_HANDLE *    ChildHandleBuffer,
                               IN UINT8           IpVersion
                             )
```

Destroy a TCP6 or TCP4 service binding instance. It will release all the resources allocated by the instance.

Parameters:

[in]	*Controller*	Controller handle of device to bind driver to.
[in]	*ImageHandle*	The TCP driver's image handle.
[in]	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If number of children is zero stop the entire bus driver.
[in]	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.
[in]	*IpVersion*	IP_VERSION_4 or IP_VERSION_6

Return values:

EFI_SUCCESS	The resources used by the instance were cleaned up.
Others	Failed to clean up some of the resources.

References TCP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer, _TCP_SERVICE_DATA::IpIo, TCP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren, TCP_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding, _TCP_SERVICE_DATA::SocketList, TCP_SERVICE_FROM_THIS, TcpDestroyChildEntryInHandleBuffer(), and TcpDestroyTimer().

Referenced by Tcp4DriverBindingStop(), and Tcp6DriverBindingStop().

```
VOID TcpDestroySocketCallback ( IN SOCKET *    This,
                                IN VOID *      Context
                              )
```

The callback function called before the TCP socket was to be destroyed.

Parameters:

[in]	*This*	The TCP socket to be destroyed.
[in]	*Context*	The context of the socket.

References _TCP_SERVICE_DATA::DriverBindingHandle, and _TCP_SERVICE_DATA::IpIo.

VOID TcpDestroyTimer (VOID)

Stop and destroy the heartbeat timer for TCP driver.

References _TCP_HEARTBEAT_TIMER::RefCnt, and _TCP_HEARTBEAT_TIMER::TimerEvent.

Referenced by TcpCreateService(), and TcpDestroyService().

EFI_STATUS EFIAPI TcpDriverEntryPoint (IN EFI_HANDLE *ImageHandle*,
IN EFI_SYSTEM_TABLE * *SystemTable*
)

The entry point for Tcp driver, which is used to install Tcp driver on the ImageHandle.

Parameters:

[in]	*ImageHandle*	The firmware allocated handle for this driver image.
[in]	*SystemTable*	Pointer to the EFI system table.

Return values:

EFI_SUCCESS	The driver loaded.
other	The driver did not load.

References gTcp4DriverBinding, gTcp6DriverBinding, gTcpComponentName, gTcpComponentName2, mTcp4RandomPort, mTcp6RandomPort, mTcpGlobalIss, and TCP_PORT_KNOWN.

EFI_STATUS EFIAPI TcpServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL * *This*,
IN OUT EFI_HANDLE * *ChildHandle*
)

Creates a child handle with a set of TCP services.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

[in]	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
[in,out]	*ChildHandle*	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.

EFI_INVALID_PARAMETER	ChildHandle is NULL.
EFI_OUT_OF_RESOURCES	There are not enough resources availabe to create the child.
other	The child handle was not created.

References _SOCK_INIT_DATA::DataSize, _SOCK_INIT_DATA::DriverBinding, _TCP_SERVICE_DATA::DriverBindingHandle, gTcp4ProtocolTemplate, gTcp6ProtocolTemplate, _TCP_SERVICE_DATA::IpVersion, _SOCK_INIT_DATA::IpVersion, _SOCK_INIT_DATA::Protocol, _SOCK_INIT_DATA::ProtoData, SockCreateChild(), _TCP_SOCKET::SockHandle, TCP_SERVICE_FROM_THIS, _TCP_PROTO_DATA::TcpPcb, and _TCP_PROTO_DATA::TcpService.

```
EFI_STATUS EFIAPI
TcpServiceBindingDestroyChild
(
  IN EFI_SERVICE_BINDING_PROTOCOL *   This,
  IN EFI_HANDLE                       ChildHandle
)
```

Destroys a child handle with a set of TCP services.

The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

This	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
ChildHandle	Handle of the child to be destroyed.

Return values:

EFI_SUCCES	The protocol was removed from ChildHandle.
EFI_UNSUPPORTED	ChildHandle does not support the protocol that is being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
EFI_ACCESS_DENIED	The protocol could not be removed from the ChildHandle because its services are being used.
other	The child handle was not destroyed.

References gTcp4DriverBinding, gTcp6DriverBinding, SOCK_FROM_THIS, and SockDestroyChild().

Variable Documentation

EFI_DRIVER_BINDING_PROTOCOL gTcp4DriverBinding

Initial value:

```
{
  Tcp4DriverBindingSupported,
  Tcp4DriverBindingStart,
  Tcp4DriverBindingStop,
  0xa,
  NULL,
  NULL
}
```

Referenced by TcpDriverEntryPoint(), and TcpServiceBindingDestroyChild().

EFI_TCP4_PROTOCOL gTcp4ProtocolTemplate

Initial value:

```
{
 Tcp4GetModeData,
 Tcp4Configure,
 Tcp4Routes,
 Tcp4Connect,
 Tcp4Accept,
 Tcp4Transmit,
 Tcp4Receive,
 Tcp4Close,
 Tcp4Cancel,
 Tcp4Poll
}
```

Referenced by TcpServiceBindingCreateChild().

EFI_DRIVER_BINDING_PROTOCOL gTcp6DriverBinding

Initial value:

```
{
 Tcp6DriverBindingSupported,
 Tcp6DriverBindingStart,
 Tcp6DriverBindingStop,
 0xa,
 NULL,
 NULL
}
```

Referenced by TcpDriverEntryPoint(), and TcpServiceBindingDestroyChild().

EFI_TCP6_PROTOCOL gTcp6ProtocolTemplate

Initial value:

```
{
 Tcp6GetModeData,
 Tcp6Configure,
 Tcp6Connect,
 Tcp6Accept,
 Tcp6Transmit,
 Tcp6Receive,
 Tcp6Close,
 Tcp6Cancel,
 Tcp6Poll
}
```

Referenced by TcpServiceBindingCreateChild().

EFI_SERVICE_BINDING_PROTOCOL gTcpServiceBinding

Initial value:

```
{
 TcpServiceBindingCreateChild,
 TcpServiceBindingDestroyChild
}
```

Referenced by TcpCreateService().

UINT16 mTcp4RandomPort

Referenced by TcpBind(), and TcpDriverEntryPoint().

UINT16 mTcp6RandomPort

Referenced by TcpBind(), and TcpDriverEntryPoint().

SOCK_INIT_DATA mTcpDefaultSockData

Initial value:

```
{
 SockStream,
 SO_CLOSED,
 NULL,
 TCP_BACKLOG,
 TCP_SND_BUF_SIZE,
 TCP_RCV_BUF_SIZE,
 IP_VERSION_4,
 NULL,
 TcpCreateSocketCallback,
 TcpDestroySocketCallback,
 NULL,
 NULL,
 0,
 TcpDispatcher,
 NULL,
}
```

TCP_HEARTBEAT_TIMER mTcpTimer

Initial value:

```
{
 NULL,
 0
}
```

NetworkPkg/TcpDxe/TcpDriver.h File Reference

Data Structures

struct _TCP_HEART
struct _TCP_SERVI
struct _TCP_PROTO

Defines

"memItemRight" valign="bottom">TCP_DRIVER_SIGNATURE SIGNATURE_32 ('T', 'C', 'P', 'D')
#define TCP_PORT_K
#define TCP_PORT_U
#define TCP_SERVIC

Typedefs

typedef struct _TCP_HEARTBEAT_TIMER TCP_HEARTI
typedef struct _TCP_SERVICE_DATA TCP_SERVIC
typedef struct _TCP_PROTO_DATA TCP_PROTO_

Functions

EFI_STATUS EFIAPI TcpDriverEntr
ImageHandle,
*SystemTable)
EFI_STATUS EFIAPI Tcp4DriverBin
EFI_DRIVER_
*This, IN EFI_
ControllerHan
EFI_DEVICE_
*RemainingDe
EFI_STATUS EFIAPI Tcp4DriverBin
EFI_DRIVER_
*This, IN EFI_
ControllerHan
EFI_DEVICE_
*RemainingDe
EFI_STATUS EFIAPI Tcp4DriverBin
EFI_DRIVER_
*This, IN EFI_
ControllerHan
NumberOfChil
*ChildHandleB
EFI_STATUS EFIAPI Tcp6DriverBin
EFI_DRIVER_
*This, IN EFI_
ControllerHan
EFI_DEVICE_
*RemainingDe

EFI_STATUS EFIAPI Tcp6DriverBindingSta
EFI_DRIVER_BINDI
*This, IN EFI_HAND
ControllerHandle, IN
EFI_DEVICE_PATH_
*RemainingDevicePat
EFI_STATUS EFIAPI Tcp6DriverBindingSto
EFI_DRIVER_BINDI
*This, IN EFI_HAND
ControllerHandle, IN U
NumberOfChildren, IN
*ChildHandleBuffer)
EFI_STATUS TcpCreateSocketCallb
*This, IN VOID *Cont
VOID TcpDestroySocketCall
*This, IN VOID *Cont
EFI_STATUS EFIAPI TcpServiceBindingCre
EFI_SERVICE_BIND
*This, IN OUT EFI_H
*ChildHandle)
EFI_STATUS EFIAPI TcpServiceBindingDes
EFI_SERVICE_BIND
*This, IN EFI_HAND

Detailed Description

The prototype of driver binding and service binding protocol for TCP driver.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by TcpCreateService().

#define TCP_PORT_KNOWN 1024

Referenced by TcpBind(), and TcpDriverEntryPoint().

#define TCP_PORT_USER_RESERVED 65535

Referenced by TcpBind().

#define TCP_SERVICE_FROM_THIS (a)

Value:

```
CR ( \
  (a), \
  TCP_SERVICE_DATA, \
  ServiceBinding, \
  TCP_DRIVER_SIGNATURE \
  )
```

Referenced by TcpDestroyService(), and TcpServiceBindingCreateChild().

Typedef Documentation

typedef struct _TCP_HEARTBEAT_TIMER TCP_HEARTBEAT_TIMER

typedef struct _TCP_PROTO_DATA TCP_PROTO_DATA

typedef struct _TCP_SERVICE_DATA TCP_SERVICE_DATA

Function Documentation

EFI_STATUS EFIAPI Tcp4DriverBindingStart (	IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*
)		

Start this driver on ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to bind driver to.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	The driver was added to ControllerHandle.
EFI_OUT_OF_RESOURCES	There are not enough resources to start the driver.
other	The driver cannot be added to ControllerHandle.

Start this driver on ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to bind driver to.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	The driver is added to ControllerHandle.
EFI_OUT_OF_RESOURCES	There are not enough resources to start the driver.

other	The driver cannot be added to ControllerHandle.

References TcpCreateService().

```
EFI_STATUS EFIAPI
Tcp4DriverBindingStop (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     ControllerHandle,
  IN UINTN                          NumberOfChildren,
  IN EFI_HANDLE *                   ChildHandleBuffer
)
```

Stop this driver on ControllerHandle.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
[in]	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.
[in]	*ChildHandleBuffer*	An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS	The device was stopped.
EFI_DEVICE_ERROR	The device could not be stopped due to a device error.

References TcpDestroyService().

```
EFI_STATUS EFIAPI
Tcp4DriverBindingSupported (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
)
```

Test to see if this driver supports ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of the device to test.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	This driver supports this device.
EFI_ALREADY_STARTED	This driver is already running on this device.
other	This driver does not support this device.

Test to see if this driver supports ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to test.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	This driver supports this device.
EFI_ALREADY_STARTED	This driver is already running on this device.
other	This driver does not support this device.

```
EFI_STATUS EFIAPI
Tcp6DriverBindingStart (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
)
```

Start this driver on ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to bind driver to.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	The driver was added to ControllerHandle.
EFI_OUT_OF_RESOURCES	There are not enough resources to start the driver.
other	The driver cannot be added to ControllerHandle.

Start this driver on ControllerHandle.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to bind driver to.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS	The driver is added to ControllerHandle.
EFI_OUT_OF_RESOURCES	There are not enough resources to start the driver.
other	The driver cannot be added to ControllerHandle.

References TcpCreateService().

```
EFI_STATUS EFIAPI
Tcp6DriverBindingStop (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     ControllerHandle,
  IN UINTN                          NumberOfChildren,
  IN EFI_HANDLE *                   ChildHandleBuffer
)
```

Stop this driver on ControllerHandle.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	A handle to the device being stopped. The handle must support a bus specific I/O protocol for the driver to use to stop the device.
[in]	*NumberOfChildren*	The number of child device handles in ChildHandleBuffer.

[in] *ChildHandleBuffer* An array of child handles to be freed. May be NULL if NumberOfChildren is 0.

Return values:

EFI_SUCCESS The device was stopped.
EFI_DEVICE_ERROR The device could not be stopped due to a device error.

References TcpDestroyService().

```
EFI_STATUS EFIAPI
Tcp6DriverBindingSupported ( IN EFI_DRIVER_BINDING_PROTOCOL * This,
                             IN EFI_HANDLE ControllerHandle,
                             IN EFI_DEVICE_PATH_PROTOCOL * RemainingDevicePath
                             )
```

Test to see if this driver supports ControllerHandle.

Parameters:

[in] *This* Protocol instance pointer.
[in] *ControllerHandle* Handle of the device to test.
[in] *RemainingDevicePath* Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS This driver supports this device.
EFI_ALREADY_STARTED This driver is already running on this device.
other This driver does not support this device.

Test to see if this driver supports ControllerHandle.

Parameters:

[in] *This* Protocol instance pointer.
[in] *ControllerHandle* Handle of device to test.
[in] *RemainingDevicePath* Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCESS This driver supports this device.
EFI_ALREADY_STARTED This driver is already running on this device.
other This driver does not support this device.

```
EFI_STATUS TcpCreateSocketCallback ( IN SOCKET * This,
                                     IN VOID * Context
                                     )
```

The Callback funtion called after the TCP socket is created.

Parameters:

[in] *This* Pointer to the socket just created.
[in] *Context* The context of the socket.

Return values:

EFI_SUCCESS This protocol is installed successfully.
other An error occured.

The Callback funtion called after the TCP socket was created.

Parameters:

[in]	*This*	Pointer to the socket just created
[in]	*Context*	Context of the socket

Return values:

EFI_SUCCESS	This protocol installed successfully.
other	An error occured.

References _TCP_SERVICE_DATA::ControllerHandle, _TCP_SERVICE_DATA::DriverBindingHandle, _TCP_SERVICE_DATA::IpIo, and _TCP_SERVICE_DATA::SocketList.

```
VOID TcpDestroySocketCallback ( IN SOCKET *   This,
                                IN VOID *     Context
                              )
```

The callback function called before the TCP socket is to be destroyed.

Parameters:

[in]	*This*	The TCP socket to be destroyed.
[in]	*Context*	The context of the socket.

The callback function called before the TCP socket was to be destroyed.

Parameters:

[in]	*This*	The TCP socket to be destroyed.
[in]	*Context*	The context of the socket.

References _TCP_SERVICE_DATA::DriverBindingHandle, and _TCP_SERVICE_DATA::IpIo.

```
EFI_STATUS EFIAPI TcpDriverEntryPoint ( IN EFI_HANDLE            ImageHandle,
                                        IN EFI_SYSTEM_TABLE *    SystemTable
                                      )
```

The entry point for Tcp driver, used to install Tcp driver on the ImageHandle.

Parameters:

[in]	*ImageHandle*	The firmware allocated handle for this driver image.
[in]	*SystemTable*	Pointer to the EFI system table.

Return values:

EFI_SUCCESS	The driver loaded.
other	The driver did not load.

The entry point for Tcp driver, which is used to install Tcp driver on the ImageHandle.

Parameters:

[in]	*ImageHandle*	The firmware allocated handle for this driver image.
[in]	*SystemTable*	Pointer to the EFI system table.

Return values:

EFI_SUCCESS	The driver loaded.
other	The driver did not load.

References gTcp4DriverBinding, gTcp6DriverBinding, gTcpComponentName, gTcpComponentName2, mTcp4RandomPort, mTcp6RandomPort, mTcpGlobalIss, and TCP_PORT_KNOWN.

```
EFI_STATUS EFIAPI
TcpServiceBindingCreateChild
(
  IN EFI_SERVICE_BINDING_PROTOCOL *   This,
  IN OUT EFI_HANDLE *                 ChildHandle
)
```

Creates a child handle with a set of TCP services.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

`[in]`	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
`[in,out]`	*ChildHandle*	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.
EFI_INVALID_PARAMETER	ChildHandle is NULL.
EFI_OUT_OF_RESOURCES	There are not enough resources available to create the child.
other	The child handle was not created.

Creates a child handle with a set of TCP services.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

`[in]`	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
`[in,out]`	*ChildHandle*	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.
EFI_INVALID_PARAMETER	ChildHandle is NULL.
EFI_OUT_OF_RESOURCES	There are not enough resources availabe to create the child.
other	The child handle was not created.

References _SOCK_INIT_DATA::DataSize, _SOCK_INIT_DATA::DriverBinding, _TCP_SERVICE_DATA::DriverBindingHandle, gTcp4ProtocolTemplate, gTcp6ProtocolTemplate, _TCP_SERVICE_DATA::IpVersion, _SOCK_INIT_DATA::IpVersion, _SOCK_INIT_DATA::Protocol, _SOCK_INIT_DATA::ProtoData, SockCreateChild(), _TCP_SOCKET::SockHandle, TCP_SERVICE_FROM_THIS, _TCP_PROTO_DATA::TcpPcb, and _TCP_PROTO_DATA::TcpService.

EFI_STATUS EFIAPI TcpServiceBindingDestroyChild	(IN EFI_SERVICE_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ChildHandle*
	)	

Destroys a child handle with a set of TCP services.

The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

This	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
ChildHandle	Handle of the child to destroy.

Return values:

EFI_SUCCES	The protocol was removed from ChildHandle.
EFI_UNSUPPORTED	ChildHandle does not support the protocol that is being removed.
EFI_INVALID_PARAMETER	The child handle is NULL.
EFI_ACCESS_DENIED	The protocol could not be removed from the ChildHandle because its services are being used.
other	The child handle was not destroyed.

Destroys a child handle with a set of TCP services.

The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

This	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
ChildHandle	Handle of the child to be destroyed.

Return values:

EFI_SUCCES	The protocol was removed from ChildHandle.
EFI_UNSUPPORTED	ChildHandle does not support the protocol that is being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
EFI_ACCESS_DENIED	The protocol could not be removed from the ChildHandle because its services are being used.
other	The child handle was not destroyed.

References gTcp4DriverBinding, gTcp6DriverBinding, SOCK_FROM_THIS, and SockDestroyChild().

NetworkPkg/TcpDxe/TcpFunc.h File Reference

Defines

"memItemRight" valign="bottom">TCP_COMP_VAL(Min, Max, Default, Val) ((((Val) <= (Max)) && ((Val) >=

Typedefs

Functions

Detailed Description

Declaration of external functions shared in TCP driver.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"paramtype">Min,
Max,
Default,
Val) ((((Val) <= (Max)) && ((Val) >= (Min))) ? (Val) : (Default))

Referenced by TcpConfigurePcb().

Typedef Documentation

typedef VOID(* TCP_TIMER_HANDLER)(IN OUT TCP_CB *Tcb)
Timeout handler prototype.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

Function Documentation

```
EFI_STATUS Tcp6RefreshNeighbor ( IN TCP_CB *              Tcb,
                                 IN EFI_IP_ADDRESS *      Neighbor,
                                 IN UINT32                Timeout
                               )
```

Refresh the remote peer's Neighbor Cache State if already exists.

Parameters:

`[in]`	*Tcb*	Pointer to the TCP_CB of this TCP instance.
`[in]`	*Neighbor*	Source address of the TCP segment.
`[in]`	*Timeout*	Time in 100-ns units that this entry will remain in the neighbor cache. A value of zero means that the entry is permanent. A value of non-zero means that the entry is dynamic and will be deleted after Timeout.

Return values:

EFI_SUCCESS	Successfully updated the neighbor relationship.
EFI_NOT_STARTED	The IpIo is not configured.
EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_OUT_OF_RESOURCES	Failed to allocate some resources.
EFI_NOT_FOUND	This entry is not in the neighbor table.

Refresh the remote peer's Neighbor Cache State if already exists.

Parameters:

`[in]`	*Tcb*	Pointer to the TCP_CB of this TCP instance.
`[in]`	*Neighbor*	Source address of the TCP segment.
`[in]`	*Timeout*	Time in 100-ns units that this entry will remain in the neighbor cache. A value of zero means that the entry is permanent. A value of non-zero means that the entry is dynamic and will be deleted after Timeout.

Return values:

EFI_SUCCESS	Successfully updated the neighbor relationship.
EFI_NOT_STARTED	The IpIo is not configured.
EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_OUT_OF_RESOURCES	Failed to allocate some resources.
EFI_NOT_FOUND	This entry is not in the neighbor table.

References _TCP_SERVICE_DATA::IpIo, _TCP_SOCKET::ProtoReserved, and _TCP_PROTO_DATA::TcpService.

Referenced by TcpInput().

UINT16 TcpChecksum (IN NET_BUF * *Nbuf*,
IN UINT16 *HeadSum*
)

Compute the TCP segment's checksum.

Parameters:

[in]	*Nbuf*	Pointer to the buffer that contains the TCP segment.
[in]	*HeadSum*	The checksum value of the fixed part of pseudo header.

Returns:

The checksum value.

Referenced by TcpInput(), TcpResetConnection(), TcpSendReset(), and TcpTransmitSegment().

VOID TcpClearAllTimer (IN OUT TCP_CB * *Tcb*)

Clear all TCP timers.

Parameters:

[in,out]	*Tcb*	Pointer to the TCP_CB of this TCP instance.

References TcpUpdateTimer().

Referenced by TcpDeliverData(), and TcpInput().

VOID TcpClearTimer (IN OUT TCP_CB * *Tcb*,
IN UINT16 *Timer*
)

Clear one TCP timer.

Parameters:

[in,out]	*Tcb*	Pointer to the TCP_CB of this TCP instance.
[in]	*Timer*	The index of the timer to be cleared.

References TCP_CLEAR_TIMER, and TcpUpdateTimer().

Referenced by TcpInput().

TCP_CB* TcpCloneTcb (IN TCP_CB * *Tcb*)

Clone a TCP_CB from Tcb.

Parameters:

[in]	*Tcb*	Pointer to the TCP_CB to be cloned.

Returns:

Pointer to the new cloned TCP_CB. If NULL, an error condition occurred.

Clone a TCP_CB from Tcb.

Parameters:

`[in]` *Tcb* Pointer to the TCP_CB to be cloned.

Returns:

Pointer to the new cloned TCP_CB; if NULL, error condition occurred.

References _TCP_CONTROL_BLOCK::List, _TCP_SOCKET::ProtoReserved, _TCP_CONTROL_BLOCK::RcvQue, _TCP_CONTROL_BLOCK::Sk, _TCP_CONTROL_BLOCK::SndQue, and SockClone().

Referenced by TcpInput().

```
                          IN OUT
VOID TcpClose    (  TCP_CB    Tcb     )
                          *
```

Close the TCP connection.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References TCP_CLOSED, and TcpSetState().

Referenced by Tcp2MSLTimeout(), TcpConnectTimeout(), TcpDeliverData(), TcpFinwait2Timeout(), TcpIcmpInput(), TcpInput(), TcpKeepaliveTimeout(), TcpOnAppClose(), and TcpRexmitTimeout().

```
UINT32 TcpDataToSend  ( IN TCP_CB *   Tcb,
                        IN INTN       Force
                      )
```

Compute how much data to send.

Parameters:

`[in]` *Tcb* Pointer to the TCP_CB of this TCP instance.

`[in]` *Force* If TRUE, ignore the sender's SWS avoidance algorithm and send out data by force.

Returns:

The length of the data that can be sent. If 0, no data can be sent.

Compute how much data to send.

Parameters:

`[in]` *Tcb* Pointer to the TCP_CB of this TCP instance.

`[in]` *Force* If TRUE, to ignore the sender's SWS avoidance algorithm and send out data by force.

Returns:

The length of the data can be sent. If 0, no data can be sent.

References GET_SND_DATASIZE, TCP_CTRL_NO_NAGLE, TCP_FLG_ON, TCP_SEQ_GT,

TCP_SUB_SEQ, TCP_TIMER_ON, TCP_TIMER_REXMIT, TcpGetMaxSndNxt(), and TcpSetProbeTimer().

Referenced by TcpProbeTimeout(), and TcpToSendData().

```
EFI_STATUS TcpDispatcher ( IN SOCKET *   Sock,
                           IN UINT8      Request,
                           IN VOID *     Data
                         )
```

The procotol handler provided to the socket layer, used to dispatch the socket level requests by calling the corresponding TCP layer functions.

Parameters:

[in]	*Sock*	Pointer to the socket of this TCP instance.
[in]	*Request*	The code of this operation request.
[in]	*Data*	Pointer to the operation specific data passed in together with the operation request. This is an optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The socket request completed successfully.
other	The error status returned by the corresponding TCP layer function.

The procotol handler provided to the socket layer, which is used to dispatch the socket level requests by calling the corresponding TCP layer functions.

Parameters:

[in]	*Sock*	Pointer to the socket of this TCP instance.
[in]	*Request*	The code of this operation request.
[in]	*Data*	Pointer to the operation specific data passed in together with the operation request. This is an optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The socket request completed successfully.
other	The error status returned by the corresponding TCP layer function.

References _TCP_CONTROL_BLOCK::CtrlFlag, GET_SND_DATASIZE, _TCP_SERVICE_DATA::IpIo, _TCP_SOCKET::IpVersion, _TCP_CONTROL_BLOCK::Sk, _TCP_CONTROL_BLOCK::SndPsh, _TCP_CONTROL_BLOCK::SndUp, SOCK_ABORT, SOCK_ATTACH, SOCK_CLOSE, SOCK_CONFIGURE, SOCK_CONNECT, SOCK_CONSUMED, SOCK_DETACH, SOCK_FLUSH, SOCK_MODE, SOCK_POLL, SOCK_ROUTE, SOCK_SND, SOCK_SNDPUSH, SOCK_SNDURG, Tcp4GetMode(), Tcp4Route(), Tcp6GetMode(), TCP_CTRL_SND_PSH, TCP_CTRL_SND_URG, TCP_SET_FLG, TcpAttachPcb(), TcpConfigurePcb(), TcpDetachPcb(), TcpFlushPcb(), TcpGetMaxSndNxt(), TcpOnAppAbort(), TcpOnAppClose(), TcpOnAppConnect(), TcpOnAppConsume(), TcpOnAppSend(), _TCP_PROTO_DATA::TcpPcb, and _TCP_PROTO_DATA::TcpService.

```
BOOLEAN TcpFindTcbByPeer ( IN EFI_IP_ADDRESS *  Addr,
                           IN TCP_PORTNO        Port,
                           IN UINT8             Version
                         )
```

Try to find one Tcb whose <Ip, Port> equals to <IN Addr, IN Port>.

Parameters:

[in] *Addr* Pointer to the IP address needs to match.
[in] *Port* The port number needs to match.
[in] *Version* IP_VERSION_4 indicates TCP is running on IP4 stack. IP_VERSION_6 indicates TCP is running on IP6 stack.

Return values:

TRUE The Tcb which matches the <Addr port>=""> pairs exists.
FALSE Otherwise

Try to find one Tcb whose <Ip, Port> equals to <IN Addr, IN Port>.

Parameters:

[in] *Addr* Pointer to the IP address needs to match.
[in] *Port* The port number needs to match.
[in] *Version* IP_VERSION_4 indicates TCP is running on IP4 stack, IP_VERSION_6 indicates TCP is running on IP6 stack.

Return values:

TRUE The Tcb which matches the <Addr port>=""> pair exists.
FALSE Otherwise

References _TCP_PEER::Ip, _TCP_SOCKET::IpVersion, _TCP_CONTROL_BLOCK::LocalEnd, mTcpListenQue, mTcpRunQue, _TCP_PEER::Port, _TCP_CONTROL_BLOCK::Sk, and TcpIsIpEqual().

Referenced by TcpBind().

```
TCP_SEG* TcpFormatNetbuf ( IN TCP_CB *          Tcb,
                           IN OUT NET_BUF *     Nbuf
                         )
```

Translate the information from the head of the received TCP segment Nbuf contains, and fill it into a TCP_SEG structure.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.
[in,out] *Nbuf* Pointer to the buffer contains the TCP segment.

Returns:

Pointer to the TCP_SEG that contains the translated TCP head information.

Translate the information from the head of the received TCP segment Nbuf contents and fill it into a TCP_SEG structure.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.
[in,out] *Nbuf* Pointer to the buffer contains the TCP segment.

Returns:

Pointer to the TCP_SEG that contains the translated TCP head information.

References _TCP_SEG::Ack, _TCP_SEG::End, _TCP_SEG::Flag, _TCP_SEG::Seq, TCP_FLG_FIN, TCP_FLG_ON, TCP_FLG_SYN, TCPSEG_NETBUF, _TCP_SEG::Urg, and _TCP_SEG::Wnd.

Referenced by TcpInput().

TCP_SEQNO TcpGetIss (VOID)

Compute an ISS to be used by a new connection.

Returns:
The result ISS.

Compute an ISS to be used by a new connection.

Returns:
The resulting ISS.

References mTcpGlobalIss, and TCP_ISS_INCREMENT_1.

Referenced by TcpInitTcbLocal().

TCP_SEQNO TcpGetMaxSndNxt (IN TCP_CB * *Tcb*)

Get the maximum SndNxt.

Parameters:
`[in]` *Tcb* Pointer to the TCP_CB of this TCP instance.

Returns:
The sequence number of the maximum SndNxt.

References TCP_SEQ_GEQ, and TCPSEG_NETBUF.

Referenced by TcpDataToSend(), TcpDispatcher(), TcpGetSegment(), and TcpToSendData().

UINT16 TcpGetRcvMss (IN SOCKET * *Sock*)

Get the local mss.

Parameters:
`[in]` *Sock* Pointer to the socket to get mss.

Returns:
The mss size.

References _TCP_SERVICE_DATA::IpIo, and _TCP_PROTO_DATA::TcpService.

Referenced by TcpConfigurePcb(), and TcpInitTcbPeer().

```
VOID TcpIcmpInput ( IN NET_BUF *            Nbuf,
                    IN UINT8                IcmpErr,
                    IN EFI_IP_ADDRESS *     Src,
                    IN EFI_IP_ADDRESS *     Dst,
                    IN UINT8                Version
                  )
```

Process the received ICMP error messages for TCP.

Parameters:

[in]	*Nbuf*	Buffer that contains part of the TCP segment without IP header truncated from the ICMP error packet.
[in]	*IcmpErr*	The ICMP error code interpreted from an ICMP error packet.
[in]	*Src*	Source address of the ICMP error message.
[in]	*Dst*	Destination address of the ICMP error message.
[in]	*Version*	IP_VERSION_4 indicates IP4 stack, IP_VERSION_6 indicates IP6 stack.

Process the received ICMP error messages for TCP.

Parameters:

[in]	*Nbuf*	The buffer that contains part of the TCP segment without an IP header truncated from the ICMP error packet.
[in]	*IcmpErr*	The ICMP error code interpreted from an ICMP error packet.
[in]	*Src*	Source address of the ICMP error message.
[in]	*Dst*	Destination address of the ICMP error message.
[in]	*Version*	IP_VERSION_4 indicates IP4 stack. IP_VERSION_6 indicates IP6 stack.

References _TCP_SOCKET::IpVersion, _TCP_CONTROL_BLOCK::Sk, _TCP_CONTROL_BLOCK::SndNxt, _TCP_CONTROL_BLOCK::SndUna, SOCK_ERROR, _TCP_CONTROL_BLOCK::State, TCP_CLOSED, TCP_SEQ_LEQ, TCP_SEQ_LT, TcpClose(), and TcpLocateTcb().

Referenced by TcpRxCallback().

VOID TcpInitTcbLocal (IN OUT TCP_CB * *Tcb*)

Initialize the Tcb locally related members.

Parameters:

[in,out]	*Tcb*	Pointer to the TCP_CB of this TCP instance.

Initialize the Tcb local related members.

Parameters:

[in,out]	*Tcb*	Pointer to the TCP_CB of this TCP instance.

References GET_RCV_BUFFSIZE, and TcpGetIss().

Referenced by TcpInput(), and TcpOnAppConnect().

VOID TcpInitTcbPeer (IN OUT TCP_CB * *Tcb*,
IN TCP_SEG * *Seg*,
IN TCP_OPTION * *Opt*
)

Initialize the peer related members.

Parameters:

[in,out]	*Tcb*	Pointer to the TCP_CB of this TCP instance.

[in]	*Seg*	Pointer to the segment that contains the peer's intial information.
[in]	*Opt*	Pointer to the options announced by the peer.

Initialize the peer related members.

Parameters:

[in,out]	*Tcb*	Pointer to the TCP_CB of this TCP instance.
[in]	*Seg*	Pointer to the segment that contains the peer's intial info.
[in]	*Opt*	Pointer to the options announced by the peer.

References TCP_CTRL_NO_TS, TCP_CTRL_NO_WS, TCP_CTRL_RCVD_TS, TCP_CTRL_RCVD_WS, TCP_CTRL_SND_TS, TCP_FLG_ACK, TCP_FLG_ON, TCP_FLG_SYN, TCP_OPTION_RCVD_MSS, TCP_OPTION_RCVD_TS, TCP_OPTION_RCVD_WS, TCP_OPTION_TS_ALIGNED_LEN, TCP_SET_FLG, TcpComputeScale(), and TcpGetRcvMss().

Referenced by TcpInput().

```
INTN TcpInput ( IN NET_BUF *          Nbuf,
                IN EFI_IP_ADDRESS *   Src,
                IN EFI_IP_ADDRESS *   Dst,
                IN UINT8              Version
              )
```

Process the received TCP segments.

Parameters:

[in]	*Nbuf*	Buffer that contains received TCP segment without an IP header.
[in]	*Src*	Source address of the segment, or the peer's IP address.
[in]	*Dst*	Destination address of the segment, or the local end's IP address.
[in]	*Version*	IP_VERSION_4 indicates IP4 stack, IP_VERSION_6 indicates IP6 stack.

Return values:

0 The segment processed successfully. It is either accepted or discarded. But no connection is reset by the segment.

-1 A connection is reset by the segment.

Process the received TCP segments.

Parameters:

[in]	*Nbuf*	Buffer that contains received a TCP segment without an IP header.
[in]	*Src*	Source address of the segment, or the peer's IP address.
[in]	*Dst*	Destination address of the segment, or the local end's IP address.
[in]	*Version*	IP_VERSION_4 indicates IP4 stack. IP_VERSION_6 indicates IP6 stack.

Return values:

0 Segment processed successfully. It is either accepted or discarded. However, no connection is reset by the segment.

-1 A connection is reset by the segment.

References _TCP_SEG::Ack, _TCP_CONTROL_BLOCK::CongestState, _TCP_CONTROL_BLOCK::ConnectTimeout, _TCP_CONTROL_BLOCK::CtrlFlag, _TCP_CONTROL_BLOCK::CWnd, _TCP_CONTROL_BLOCK::DupAck, _TCP_SEG::End,

_TCP_CONTROL_BLOCK::FinWait2Timeout, _TCP_OPTION::Flag, _TCP_SEG::Flag, _TCP_CONTROL_BLOCK::Idle, _TCP_PEER::Ip, _TCP_CONTROL_BLOCK::IpInfo, _TCP_CONTROL_BLOCK::Iss, _TCP_CONTROL_BLOCK::LocalEnd, _TCP_CONTROL_BLOCK::LossRecover, mTcpTick, _TCP_CONTROL_BLOCK::Parent, _TCP_PEER::Port, _TCP_CONTROL_BLOCK::ProbeTimerOn, _TCP_CONTROL_BLOCK::RcvQue, _TCP_CONTROL_BLOCK::RcvUp, _TCP_CONTROL_BLOCK::RcvWl2, _TCP_CONTROL_BLOCK::Recover, _TCP_CONTROL_BLOCK::RemoteEnd, _TCP_CONTROL_BLOCK::Rto, _TCP_CONTROL_BLOCK::RttMeasure, _TCP_SEG::Seq, _TCP_CONTROL_BLOCK::Sk, _TCP_CONTROL_BLOCK::SndMss, _TCP_CONTROL_BLOCK::SndNxt, _TCP_CONTROL_BLOCK::SndUna, _TCP_CONTROL_BLOCK::SndUp, _TCP_CONTROL_BLOCK::SndWl1, _TCP_CONTROL_BLOCK::SndWl2, _TCP_CONTROL_BLOCK::SndWnd, _TCP_CONTROL_BLOCK::SndWndMax, _TCP_CONTROL_BLOCK::SndWndScale, SOCK_ERROR, _TCP_CONTROL_BLOCK::Ssthresh, _TCP_CONTROL_BLOCK::State, TCP6_KEEP_NEIGHBOR_TIME, TCP6_REFRESH_NEIGHBOR_TICK, Tcp6RefreshNeighbor(), TCP_CLEAR_FLG, TCP_CLOSE_WAIT, TCP_CLOSED, TCP_CLOSING, TCP_CONGEST_LOSS, TCP_CONGEST_OPEN, TCP_CONGEST_RECOVER, TCP_CTRL_ACK_NOW, TCP_CTRL_FIN_ACKED, TCP_CTRL_FIN_SENT, TCP_CTRL_RCVD_URG, TCP_CTRL_RTT_ON, TCP_CTRL_SND_URG, TCP_ESTABLISHED, TCP_FIN_RCVD, TCP_FIN_WAIT_1, TCP_FIN_WAIT_2, TCP_FLG_ACK, TCP_FLG_FIN, TCP_FLG_ON, TCP_FLG_RST, TCP_FLG_SYN, TCP_FLG_URG, TCP_LAST_ACK, TCP_LISTEN, TCP_LOCAL_CLOSED, TCP_MAX_WIN, TCP_OPTION_RCVD_TS, TCP_SEQ_GT, TCP_SEQ_LEQ, TCP_SEQ_LT, TCP_SET_FLG, TCP_SUB_TIME, TCP_SYN_RCVD, TCP_SYN_SENT, TCP_TIME_WAIT, TCP_TIMER_2MSL, TCP_TIMER_CONNECT, TCP_TIMER_FINWAIT2, TCP_TIMER_PROBE, TCP_TIMER_REXMIT, TcpAdjustSndQue(), TcpChecksum(), TcpClearAllTimer(), TcpClearTimer(), TcpCloneTcb(), TcpClose(), TcpComputeRtt(), TcpDeliverData(), TcpFastLossRecover(), TcpFastRecover(), TcpFormatNetbuf(), TcpInitTcbLocal(), TcpInitTcbPeer(), TcpInsertTcb(), TcpLocateTcb(), TcpParseOption(), TcpQueueData(), TcpSendAck(), TcpSendReset(), TcpSeqAcceptable(), TcpSetKeepaliveTimer(), TcpSetProbeTimer(), TcpSetState(), TcpSetTimer(), TcpToSendAck(), TcpToSendData(), TcpTrimInWnd(), _TCP_CONTROL_BLOCK::Tick, _TCP_CONTROL_BLOCK::TimeWaitTimeout, _TCP_OPTION::TSEcr, _TCP_CONTROL_BLOCK::TsRecent, _TCP_CONTROL_BLOCK::TsRecentAge, _TCP_OPTION::TSVal, _TCP_SEG::Urg, and _TCP_SEG::Wnd.

Referenced by TcpRxCallback().

INTN TcpInsertTcb (IN TCP_CB * *Tcb*)

Insert a Tcb into the proper queue.

Parameters:

`[in]` *Tcb* Pointer to the TCP_CB to be inserted.

Return values:

0 The Tcb was inserted successfully.

-1 An error condition occurred.

Insert a Tcb into the proper queue.

Parameters:

`[in]` *Tcb* Pointer to the TCP_CB to be inserted.

Return values:

0 The Tcb was inserted successfully.

-1 Error condition occurred.

References _TCP_CONTROL_BLOCK::LocalEnd, mTcpListenQue, mTcpRunQue, _TCP_CONTROL_BLOCK::RemoteEnd, TCP_CLOSED, TCP_LISTEN, TCP_PEER_EQUAL, TCP_SYN_RCVD, and TCP_SYN_SENT.

Referenced by TcpConfigurePcb(), and TcpInput().

EFI_STATUS TcpInstallDevicePath (IN SOCKET * *Sock*)

Install the device path protocol on the TCP instance.

Parameters:

[in] *Sock* Pointer to the socket representing the TCP instance.

Return values:

EFI_SUCCESS The device path protocol installed.
other Failed to install the device path protocol.

Install the device path protocol on the TCP instance.

Parameters:

[in] *Sock* Pointer to the socket representing the TCP instance.

Return values:

EFI_SUCCESS The device path protocol was installed.
other Failed to install the device path protocol.

References _TCP_SERVICE_DATA::ControllerHandle, _TCP_PEER::Ip, _TCP_CONTROL_BLOCK::LocalEnd, _TCP_PEER::Port, _TCP_CONTROL_BLOCK::RemoteEnd, _TCP_CONTROL_BLOCK::SubnetMask, _TCP_PROTO_DATA::TcpPcb, _TCP_PROTO_DATA::TcpService, and _TCP_CONTROL_BLOCK::UseDefaultAddr.

Referenced by TcpConfigurePcb(), and TcpSetState().

TCP_CB* TcpLocateTcb (IN TCP_PORTNO *LocalPort*,
IN EFI_IP_ADDRESS * *LocalIp*,
IN TCP_PORTNO *RemotePort*,
IN EFI_IP_ADDRESS * *RemoteIp*,
IN UINT8 *Version*,
IN BOOLEAN *Syn*
)

Locate the TCP_CB related to the socket pair.

Parameters:

[in] *LocalPort* The local port number.
[in] *LocalIp* The local IP address.
[in] *RemotePort* The remote port number.
[in] *RemoteIp* The remote IP address.
[in] *Version* IP_VERSION_4 indicates TCP is running on IP4 stack, IP_VERSION_6 indicates TCP is running on IP6 stack.
[in] *Syn* If TRUE, the listen sockets are searched.

Returns:
Pointer to the related TCP_CB. If NULL, no match is found.

References _TCP_PEER::Ip, _TCP_SOCKET::IpVersion, _TCP_CONTROL_BLOCK::List, _TCP_CONTROL_BLOCK::LocalEnd, mTcpRunQue, _TCP_PEER::Port, _TCP_CONTROL_BLOCK::RemoteEnd, _TCP_CONTROL_BLOCK::Sk, TCP_PEER_EQUAL, and TcpLocateListenTcb().

Referenced by TcpIcmpInput(), and TcpInput().

VOID TcpOnAppAbort (IN TCP_CB * *Tcb*)

Abort the connection by sending a reset segment: called when the application wants to abort the connection.

Parameters:
`[in]` *Tcb* Pointer to the TCP_CB of the TCP instance.

Abort the connection by sending a reset segment. Called when the application wants to abort the connection.

Parameters:
`[in]` *Tcb* Pointer to the TCP_CB of the TCP instance.

References TCP_CLOSE_WAIT, TCP_CLOSED, TCP_ESTABLISHED, TCP_FIN_WAIT_1, TCP_FIN_WAIT_2, TCP_SYN_RCVD, TcpResetConnection(), and TcpSetState().

Referenced by TcpDispatcher().

VOID TcpOnAppClose (IN OUT TCP_CB * *Tcb*)

Initiate the connection close procedure, called when applications want to close the connection.

Parameters:
`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References GET_RCV_DATASIZE, TCP_CLOSE_WAIT, TCP_CLOSED, TCP_ESTABLISHED, TCP_FIN_WAIT_1, TCP_LAST_ACK, TCP_LISTEN, TCP_SYN_RCVD, TCP_SYN_SENT, TcpClose(), TcpResetConnection(), TcpSetState(), and TcpToSendData().

Referenced by TcpDispatcher().

VOID TcpOnAppConnect (IN OUT TCP_CB * *Tcb*)

Initialize an active connection,

Parameters:
`[in,out]` *Tcb* Pointer to the TCP_CB that wants to initiate a connection.

Initialize an active connection.

Parameters:
`[in,out]` *Tcb* Pointer to the TCP_CB that wants to initiate a connection.

References TCP_SYN_SENT, TCP_TIMER_CONNECT, TcpInitTcbLocal(), TcpSetState(), TcpSetTimer(), and TcpToSendData().

Referenced by TcpDispatcher().

VOID TcpOnAppConsume (IN TCP_CB * *Tcb*)

Application has consumed some data, check whether to send a window update ack or a delayed ack.

Parameters:

`[in]` *Tcb* Pointer to the TCP_CB of this TCP instance.

Application has consumed some data. Check whether to send a window update ack or a delayed ack.

Parameters:

`[in]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References TCP_ESTABLISHED, TcpRcvWinNow(), TcpRcvWinOld(), and TcpSendAck().

Referenced by TcpDispatcher().

INTN TcpOnAppSend (IN OUT TCP_CB * *Tcb*)

Check whether the application's newly delivered data can be sent out.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

Return values:

0 The data has been sent out successfully.

-1 The Tcb is not in a state that data is permitted to be sent out.

References TCP_CLOSE_WAIT, TCP_CLOSED, TCP_CLOSING, TCP_ESTABLISHED, TCP_FIN_WAIT_1, TCP_FIN_WAIT_2, TCP_LAST_ACK, TCP_LISTEN, TCP_SYN_RCVD, TCP_SYN_SENT, TCP_TIME_WAIT, and TcpToSendData().

Referenced by TcpDispatcher().

UINT32 TcpRcvWinNow (IN TCP_CB * *Tcb*)

Compute the current receive window.

Parameters:

`[in]` *Tcb* Pointer to the TCP_CB of this TCP instance.

Returns:

The size of the current receive window, in bytes.

References GET_RCV_BUFFSIZE, SOCK_RCV_BUF, SockGetFreeSpace(), and TcpRcvWinOld().

Referenced by TcpComputeWnd(), TcpOnAppConsume(), and TcpToSendAck().

UINT32 TcpRcvWinOld (IN TCP_CB * *Tcb*)

Compute the sequence space left in the old receive window.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.

Returns:

The sequence space left in the old receive window.

References TCP_SEQ_GT, and TCP_SUB_SEQ.

Referenced by TcpOnAppConsume(), TcpRcvWinNow(), and TcpToSendAck().

VOID TcpResetConnection (IN TCP_CB * *Tcb*)

Reset the connection related with Tcb.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of the connection to be reset.

References TCP_FLG_RST, TCP_MAX_HEAD, TcpChecksum(), and TcpSendIpPacket().

Referenced by TcpConnectTimeout(), TcpOnAppAbort(), and TcpOnAppClose().

INTN TcpRetransmit (IN TCP_CB * *Tcb*,
IN TCP_SEQNO *Seq*
)

Retransmit the segment from sequence Seq.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.

[in] *Seq* The sequence number of the segment to be retransmitted.

Return values:

0 The retransmission succeeded.

-1 An error condition occurred.

Retransmit the segment from sequence Seq.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.

[in] *Seq* The sequence number of the segment to be retransmitted.

Return values:

0 Retransmission succeeded.

-1 Error condition occurred.

References TCP_SEQ_LT, TCP_SUB_SEQ, TcpGetSegmentSndQue(), TcpTransmitSegment(), and TcpVerifySegment().

Referenced by TcpFastLossRecover(), TcpFastRecover(), and TcpRexmitTimeout().

```
VOID EFIAPI TcpRxCallback ( IN EFI_STATUS                Status,
                            IN UINT8                     IcmpErr,
                            IN EFI_NET_SESSION_DATA *    NetSession,
                            IN NET_BUF *                 Pkt,
                            IN VOID *                    Context
                          )
```

Packet receive callback function provided to IP_IO. Used to call the proper function to handle the packet received by IP.

Parameters:

[in]	*Status*	Result of the receive request.
[in]	*IcmpErr*	Valid when Status is EFI_ICMP_ERROR.
[in]	*NetSession*	The IP session for the received packet.
[in]	*Pkt*	Packet received.
[in]	*Context*	The data provided by the user for the received packet when the callback is registered in IP_IO_OPEN_DATA::RcvdContext. This is an optional parameter that may be NULL.

Packet receive callback function provided to IP_IO, used to call the proper function to handle the packet received by IP.

Parameters:

[in]	*Status*	Result of the receive request.
[in]	*IcmpErr*	Valid when Status is EFI_ICMP_ERROR.
[in]	*NetSession*	The IP session for the received packet.
[in]	*Pkt*	Packet received.
[in]	*Context*	The data provided by the user for the received packet when the callback is registered in IP_IO_OPEN_DATA::RcvdContext. This is an optional parameter that may be NULL.

References TcpIcmpInput(), and TcpInput().

Referenced by TcpCreateService().

```
VOID TcpSendAck ( IN OUT TCP_CB *    Tcb    )
```

Send an ACK immediately.

Parameters:

[in,out]	*Tcb*	Pointer to the TCP_CB of this TCP instance.

References _TCP_SEG::End, _TCP_SEG::Flag, _TCP_SEG::Seq, TCP_CLEAR_FLG, TCP_CTRL_ACK_NOW, TCP_FLG_ACK, TCP_MAX_HEAD, TCPSEG_NETBUF, and TcpTransmitSegment().

Referenced by TcpDeliverData(), TcpInput(), TcpOnAppConsume(), TcpTickingDpc(), and TcpToSendAck().

```
INTN TcpSendIpPacket ( IN TCP_CB *          Tcb,
                       IN NET_BUF *         Nbuf,
```

```
                    IN EFI_IP_ADDRESS *   Src,
                    IN EFI_IP_ADDRESS *   Dest,
                    IN UINT8              Version
                  )
```

Send the segment to IP via IpIo function.

Parameters:

- [in] *Tcb* Pointer to the TCP_CB of this TCP instance.
- [in] *Nbuf* Pointer to the TCP segment to be sent.
- [in] *Src* Source address of the TCP segment.
- [in] *Dest* Destination address of the TCP segment.
- [in] *Version* IP_VERSION_4 or IP_VERSION_6

Return values:

- *0* The segment was sent out successfully.
- *-1* The segment failed to be sent.

Send the segment to IP via IpIo function.

Parameters:

- [in] *Tcb* Pointer to the TCP_CB of this TCP instance.
- [in] *Nbuf* Pointer to the TCP segment to be sent.
- [in] *Src* Source address of the TCP segment.
- [in] *Dest* Destination address of the TCP segment.
- [in] *Version* IP_VERSION_4 or IP_VERSION_6

Return values:

- *0* The segment was sent out successfully.
- *-1* The segment failed to send.

References _TCP_SERVICE_DATA::IpIo, _TCP_SOCKET::ProtoReserved, and _TCP_PROTO_DATA::TcpService.

Referenced by TcpResetConnection(), TcpSendReset(), and TcpTransmitSegment().

```
INTN TcpSendReset ( IN TCP_CB *           Tcb,
                    IN TCP_HEAD *         Head,
                    IN INT32              Len,
                    IN EFI_IP_ADDRESS *   Local,
                    IN EFI_IP_ADDRESS *   Remote,
                    IN UINT8              Version
                  )
```

Send a RESET segment in response to the segment received.

Parameters:

- [in] *Tcb* Pointer to the TCP_CB of this TCP instance, may be NULL.
- [in] *Head* TCP header of the segment that triggers the reset.
- [in] *Len* Length of the segment that triggers the reset.
- [in] *Local* Local IP address.
- [in] *Remote* Remote peer's IP address.

[in] *Version* IP_VERSION_4 indicates TCP is running on IP4 stack, IP_VERSION_6 indicates TCP is running on IP6 stack.

Return values:

0 A reset is sent or no need to send it.

-1 No reset is sent.

Send a RESET segment in response to the segment received.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance. May be NULL.

[in] *Head* TCP header of the segment that triggers the reset.

[in] *Len* Length of the segment that triggers the reset.

[in] *Local* Local IP address.

[in] *Remote* Remote peer's IP address.

[in] *Version* IP_VERSION_4 indicates TCP is running on IP4 stack, IP_VERSION_6 indicates TCP is running on IP6 stack.

Return values:

0 A reset was sent or there is no need to send it.

-1 No reset is sent.

References TCP_FLG_ACK, TCP_FLG_ON, TCP_FLG_RST, TCP_MAX_HEAD, TCP_SET_FLG, TcpChecksum(), and TcpSendIpPacket().

Referenced by TcpInput().

INTN TcpSendZeroProbe (IN OUT TCP_CB * *Tcb*)

Send a zero probe segment. It can be used by keepalive and zero window probe.

Parameters:

[in,out] *Tcb* Pointer to the TCP_CB of this TCP instance.

Return values:

0 The zero probe segment was sent out successfully.

other An error condition occurred.

References _TCP_SEG::End, _TCP_SEG::Flag, _TCP_SEG::Seq, TCP_FLG_ACK, TCP_MAX_HEAD, TCPSEG_NETBUF, and TcpTransmitSegment().

Referenced by TcpKeepaliveTimeout(), and TcpProbeTimeout().

VOID TcpSetKeepaliveTimer (IN OUT TCP_CB * *Tcb*)

Enable the keepalive timer and set the timeout value.

Parameters:

[in,out] *Tcb* Pointer to the TCP_CB of this TCP instance.

References TCP_CTRL_NO_KEEPALIVE, TCP_FLG_ON, TCP_TIMER_KEEPALIVE, TCP_TIMER_ON, and TcpSetTimer().

Referenced by TcpInput(), and TcpKeepaliveTimeout().

VOID TcpSetProbeTimer (IN OUT TCP_CB * *Tcb*)

Enable the window prober timer and set the timeout value.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References TCP_RTO_MAX, TCP_RTO_MIN, TCP_TIMER_PROBE, and TcpSetTimer().

Referenced by TcpDataToSend(), TcpInput(), and TcpProbeTimeout().

VOID TcpSetState (IN TCP_CB * *Tcb*,
IN UINT8 *State*
)

Set the Tcb's state.

Parameters:

`[in]` *Tcb* Pointer to the TCP_CB of this TCP instance.
`[in]` *State* The state to be set.

References mTcpStateName, SockConnClosed(), SockConnEstablished(), TCP_CLOSED, TCP_ESTABLISHED, and TcpInstallDevicePath().

Referenced by TcpClose(), TcpConfigurePcb(), TcpDeliverData(), TcpInput(), TcpOnAppAbort(), TcpOnAppClose(), and TcpOnAppConnect().

VOID TcpSetTimer (IN OUT TCP_CB * *Tcb*,
IN UINT16 *Timer*,
IN UINT32 *TimeOut*
)

Enable a TCP timer.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.
`[in]` *Timer* The index of the timer to be enabled.
`[in]` *TimeOut* The timeout value of this timer.

References mTcpTick, TCP_SET_TIMER, and TcpUpdateTimer().

Referenced by TcpDeliverData(), TcpInput(), TcpOnAppConnect(), TcpRexmitTimeout(), TcpSetKeepaliveTimer(), TcpSetProbeTimer(), and TcpToSendData().

VOID EFIAPI TcpTicking (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

Heart beat timer handler, queues the DPC at TPL_CALLBACK.

Parameters:

[in] *Event* Timer event signaled, ignored.
[in] *Context* Context of the timer event, ignored.

References TcpTickingDpc().

Referenced by TcpCreateTimer().

VOID TcpToSendAck (IN OUT TCP_CB * *Tcb*)

Check whether to send an ACK or delayed ACK.

Parameters:

[in,out] *Tcb* Pointer to the TCP_CB of this TCP instance.

References TCP_CTRL_ACK_NOW, TCP_FLG_ON, TcpRcvWinNow(), TcpRcvWinOld(), and TcpSendAck().

Referenced by TcpInput().

INTN TcpToSendData (IN OUT TCP_CB * *Tcb*,
IN INTN *Force*
)

Check whether to send data/SYN/FIN and piggyback an ACK.

Parameters:

[in,out] *Tcb* Pointer to the TCP_CB of this TCP instance.
[in] *Force* If TRUE, ignore the sender's SWS avoidance algorithm and send out data by force.

Returns:

The number of bytes sent.

References _TCP_SEG::End, _TCP_SEG::Flag, GET_SND_DATASIZE, mTcpOutFlag, mTcpStateName, _TCP_SEG::Seq, TCP_CLEAR_FLG, TCP_CLOSED, TCP_CONGEST_OPEN, TCP_CTRL_FIN_SENT, TCP_CTRL_RTT_ON, TCP_FLG_FIN, TCP_FLG_ON, TCP_FLG_SYN, TCP_LISTEN, TCP_SEQ_GT, TCP_SEQ_LT, TCP_SET_FLG, TCP_SUB_SEQ, TCP_TIMER_ON, TCP_TIMER_REXMIT, TcpCheckSndQue(), TcpDataToSend(), TcpGetMaxSndNxt(), TcpGetSegment(), TCPSEG_NETBUF, TcpSetTimer(), TcpTransmitSegment(), and TcpVerifySegment().

Referenced by TcpInput(), TcpOnAppClose(), TcpOnAppConnect(), TcpOnAppSend(), and TcpProbeTimeout().

INTN TcpVerifySegment (IN NET_BUF * *Nbuf*)

Verify that the segment is in good shape.

Parameters:

[in] *Nbuf* Buffer that contains the segment to be checked.

Return values:

0 The segment is broken.

1 The segment is in good shape.

Verify that the segment is in good shape.

Parameters:

`[in]` *Nbuf* The buffer that contains the segment to be checked.

Return values:

0 The segment is broken.

1 The segment is in good shape.

References _TCP_SEG::End, _TCP_SEG::Flag, _TCP_SEG::Seq, TCP_FLG_FIN, TCP_FLG_ON, TCP_FLG_SYN, and TCPSEG_NETBUF.

Referenced by TcpCheckSndQue(), TcpDeliverData(), TcpGetSegment(), TcpRetransmit(), TcpToSendData(), TcpTransmitSegment(), and TcpTrimSegment().

NetworkPkg/TcpDxe/TcpInput.c File Reference

Functions

INTN TcpSeqAcceptable (IN TCP_CB *Tcb, IN TCP_SEG *Seg)
VOID TcpFastRecover (IN OUT TCP_CB *Tcb, IN TCP_SEG *Seg)
VOID TcpFastLossRecover (IN OUT TCP_CB *Tcb, IN TCP_SEG *Seg)
VOID TcpComputeRtt (IN OUT TCP_CB *Tcb, IN UINT32 Measure)
VOID TcpTrimSegment (IN NET_BUF *Nbuf, IN TCP_SEQNO Left, IN TCP_SEQNO Right)
VOID TcpTrimInWnd (IN TCP_CB *Tcb, IN NET_BUF *Nbuf)
INTN TcpDeliverData (IN OUT TCP_CB *Tcb)
VOID TcpQueueData (IN OUT TCP_CB *Tcb, IN NET_BUF *Nbuf)
VOID TcpAdjustSndQue (IN TCP_CB *Tcb, IN TCP_SEQNO Ack)
INTN TcpInput (IN NET_BUF *Nbuf, IN EFI_IP_ADDRESS *Src, IN EFI_IP_ADDRESS *Dst, IN UINT8 Version)
VOID TcpIcmpInput (IN NET_BUF *Nbuf, IN UINT8 IcmpErr, IN EFI_IP_ADDRESS *Src, IN EFI_IP_ADDRESS *Dst, IN UINT8 Version)

Detailed Description

TCP input process routines.

Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
VOID TcpAdjustSndQue ( IN TCP_CB *      Tcb,
                       IN TCP_SEQNO     Ack
                     )
```

Adjust the send queue or the retransmit queue.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.
[in] *Ack* The acknowledge seuqence number of the received segment.

References _TCP_SEG::End, _TCP_SEG::Seq, TCP_SEQ_GEQ, TCP_SEQ_LEQ, TCPSEG_NETBUF, and TcpTrimSegment().

Referenced by TcpInput().

VOID TcpComputeRtt (IN OUT TCP_CB * *Tcb*,
IN UINT32 *Measure*
)

Compute the RTT as specified in RFC2988.

Parameters:

[in,out]	*Tcb*	Pointer to the TCP_CB of this TCP instance.
[in]	*Measure*	Currently measured RTT in heartbeats.

References TCP_RTO_MAX, TCP_RTO_MIN, and TCP_RTT_SHIFT.

Referenced by TcpInput().

INTN TcpDeliverData (IN OUT TCP_CB * *Tcb*)

Process the data and FIN flag, and check whether to deliver data to the socket layer.

Parameters:

[in,out]	*Tcb*	Pointer to the TCP_CB of this TCP instance.

Return values:

0	No error occurred to deliver data.
-1	An error condition occurred. The proper response is to reset the connection.

References _TCP_SEG::End, _TCP_SEG::Flag, _TCP_SEG::Seq, SockDataRcvd(), SockNoMoreData(), TCP_CLOSE_WAIT, TCP_CLOSING, TCP_CONNECTED, TCP_CTRL_ACK_NOW, TCP_CTRL_FIN_ACKED, TCP_CTRL_RCVD_URG, TCP_ESTABLISHED, TCP_FIN_RCVD, TCP_FIN_WAIT_1, TCP_FIN_WAIT_2, TCP_FLG_FIN, TCP_FLG_ON, TCP_FLG_PSH, TCP_LAST_ACK, TCP_SEQ_GT, TCP_SEQ_LEQ, TCP_SET_FLG, TCP_SUB_SEQ, TCP_SYN_RCVD, TCP_TIME_WAIT, TCP_TIMER_2MSL, TcpClearAllTimer(), TcpClose(), TCPSEG_NETBUF, TcpSendAck(), TcpSetState(), TcpSetTimer(), and TcpVerifySegment().

Referenced by TcpInput().

VOID TcpFastLossRecover (IN OUT TCP_CB * *Tcb*,
IN TCP_SEG * *Seg*
)

NewReno fast loss recovery defined in RFC3792.

Parameters:

[in,out]	*Tcb*	Pointer to the TCP_CB of this TCP instance.
[in]	*Seg*	Segment that triggers the fast loss recovery.

References TCP_CONGEST_OPEN, TCP_SEQ_GEQ, TCP_SEQ_GT, and TcpRetransmit().

Referenced by TcpInput().

VOID TcpFastRecover (IN OUT TCP_CB * *Tcb*,
Seg

```
                    IN
                    TCP_SEG
                    *
                )
```

NewReno fast recovery defined in RFC3782.

Parameters:

[in,out]	*Tcb*	Pointer to the TCP_CB of this TCP instance.
[in]	*Seg*	Segment that triggers the fast recovery.

References TCP_CLEAR_FLG, TCP_CONGEST_OPEN, TCP_CONGEST_RECOVER, TCP_CTRL_RTT_ON, TCP_SEQ_GEQ, TCP_SUB_SEQ, and TcpRetransmit().

Referenced by TcpInput().

```
VOID TcpIcmpInput ( IN NET_BUF *            Nbuf,
                    IN UINT8                IcmpErr,
                    IN EFI_IP_ADDRESS *     Src,
                    IN EFI_IP_ADDRESS *     Dst,
                    IN UINT8                Version
                  )
```

Process the received ICMP error messages for TCP.

Parameters:

[in]	*Nbuf*	The buffer that contains part of the TCP segment without an IP header truncated from the ICMP error packet.
[in]	*IcmpErr*	The ICMP error code interpreted from an ICMP error packet.
[in]	*Src*	Source address of the ICMP error message.
[in]	*Dst*	Destination address of the ICMP error message.
[in]	*Version*	IP_VERSION_4 indicates IP4 stack. IP_VERSION_6 indicates IP6 stack.

References _TCP_SOCKET::IpVersion, _TCP_CONTROL_BLOCK::Sk, _TCP_CONTROL_BLOCK::SndNxt, _TCP_CONTROL_BLOCK::SndUna, SOCK_ERROR, _TCP_CONTROL_BLOCK::State, TCP_CLOSED, TCP_SEQ_LEQ, TCP_SEQ_LT, TcpClose(), and TcpLocateTcb().

Referenced by TcpRxCallback().

```
INTN TcpInput ( IN NET_BUF *            Nbuf,
                IN EFI_IP_ADDRESS *     Src,
                IN EFI_IP_ADDRESS *     Dst,
                IN UINT8                Version
              )
```

Process the received TCP segments.

Parameters:

[in]	*Nbuf*	Buffer that contains received a TCP segment without an IP header.
[in]	*Src*	Source address of the segment, or the peer's IP address.
[in]	*Dst*	Destination address of the segment, or the local end's IP address.

[in] *Version* IP_VERSION_4 indicates IP4 stack. IP_VERSION_6 indicates IP6 stack.

Return values:

0 Segment processed successfully. It is either accepted or discarded. However, no connection is reset by the segment.

-1 A connection is reset by the segment.

References _TCP_SEG::Ack, _TCP_CONTROL_BLOCK::CongestState, _TCP_CONTROL_BLOCK::ConnectTimeout, _TCP_CONTROL_BLOCK::CtrlFlag, _TCP_CONTROL_BLOCK::CWnd, _TCP_CONTROL_BLOCK::DupAck, _TCP_SEG::End, _TCP_CONTROL_BLOCK::FinWait2Timeout, _TCP_OPTION::Flag, _TCP_SEG::Flag, _TCP_CONTROL_BLOCK::Idle, _TCP_PEER::Ip, _TCP_CONTROL_BLOCK::IpInfo, _TCP_CONTROL_BLOCK::Iss, _TCP_CONTROL_BLOCK::LocalEnd, _TCP_CONTROL_BLOCK::LossRecover, mTcpTick, _TCP_CONTROL_BLOCK::Parent, _TCP_PEER::Port, _TCP_CONTROL_BLOCK::ProbeTimerOn, _TCP_CONTROL_BLOCK::RcvQue, _TCP_CONTROL_BLOCK::RcvUp, _TCP_CONTROL_BLOCK::RcvWl2, _TCP_CONTROL_BLOCK::Recover, _TCP_CONTROL_BLOCK::RemoteEnd, _TCP_CONTROL_BLOCK::Rto, _TCP_CONTROL_BLOCK::RttMeasure, _TCP_SEG::Seq, _TCP_CONTROL_BLOCK::Sk, _TCP_CONTROL_BLOCK::SndMss, _TCP_CONTROL_BLOCK::SndNxt, _TCP_CONTROL_BLOCK::SndUna, _TCP_CONTROL_BLOCK::SndUp, _TCP_CONTROL_BLOCK::SndWl1, _TCP_CONTROL_BLOCK::SndWl2, _TCP_CONTROL_BLOCK::SndWnd, _TCP_CONTROL_BLOCK::SndWndMax, _TCP_CONTROL_BLOCK::SndWndScale, SOCK_ERROR, _TCP_CONTROL_BLOCK::Ssthresh, _TCP_CONTROL_BLOCK::State, TCP6_KEEP_NEIGHBOR_TIME, TCP6_REFRESH_NEIGHBOR_TICK, Tcp6RefreshNeighbor(), TCP_CLEAR_FLG, TCP_CLOSE_WAIT, TCP_CLOSED, TCP_CLOSING, TCP_CONGEST_LOSS, TCP_CONGEST_OPEN, TCP_CONGEST_RECOVER, TCP_CTRL_ACK_NOW, TCP_CTRL_FIN_ACKED, TCP_CTRL_FIN_SENT, TCP_CTRL_RCVD_URG, TCP_CTRL_RTT_ON, TCP_CTRL_SND_URG, TCP_ESTABLISHED, TCP_FIN_RCVD, TCP_FIN_WAIT_1, TCP_FIN_WAIT_2, TCP_FLG_ACK, TCP_FLG_FIN, TCP_FLG_ON, TCP_FLG_RST, TCP_FLG_SYN, TCP_FLG_URG, TCP_LAST_ACK, TCP_LISTEN, TCP_LOCAL_CLOSED, TCP_MAX_WIN, TCP_OPTION_RCVD_TS, TCP_SEQ_GT, TCP_SEQ_LEQ, TCP_SEQ_LT, TCP_SET_FLG, TCP_SUB_TIME, TCP_SYN_RCVD, TCP_SYN_SENT, TCP_TIME_WAIT, TCP_TIMER_2MSL, TCP_TIMER_CONNECT, TCP_TIMER_FINWAIT2, TCP_TIMER_PROBE, TCP_TIMER_REXMIT, TcpAdjustSndQue(), TcpChecksum(), TcpClearAllTimer(), TcpClearTimer(), TcpCloneTcb(), TcpClose(), TcpComputeRtt(), TcpDeliverData(), TcpFastLossRecover(), TcpFastRecover(), TcpFormatNetbuf(), TcpInitTcbLocal(), TcpInitTcbPeer(), TcpInsertTcb(), TcpLocateTcb(), TcpParseOption(), TcpQueueData(), TcpSendAck(), TcpSendReset(), TcpSeqAcceptable(), TcpSetKeepaliveTimer(), TcpSetProbeTimer(), TcpSetState(), TcpSetTimer(), TcpToSendAck(), TcpToSendData(), TcpTrimInWnd(), _TCP_CONTROL_BLOCK::Tick, _TCP_CONTROL_BLOCK::TimeWaitTimeout, _TCP_OPTION::TSEcr, _TCP_CONTROL_BLOCK::TsRecent, _TCP_CONTROL_BLOCK::TsRecentAge, _TCP_OPTION::TSVal, _TCP_SEG::Urg, and _TCP_SEG::Wnd.

Referenced by TcpRxCallback().

```
VOID TcpQueueData ( IN OUT TCP_CB *    Tcb,
                    IN NET_BUF *       Nbuf
                  )
```

Store the data into the reassemble queue.

Parameters:

[in,out] *Tcb* Pointer to the TCP_CB of this TCP instance.

[in] *Nbuf* Pointer to the buffer containing the data to be queued.

References _TCP_SEG::End, _TCP_SEG::Seq, TCP_CTRL_ACK_NOW, TCP_SEQ_LEQ, TCP_SEQ_LT, TCP_SET_FLG, TCPSEG_NETBUF, and TcpTrimSegment().

Referenced by TcpInput().

```
INTN TcpSeqAcceptable ( IN TCP_CB *     Tcb,
                        IN TCP_SEG *    Seg
                      )
```

Check whether the sequence number of the incoming segment is acceptable.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.
[in] *Seg* Pointer to the incoming segment.

Return values:

1 The sequence number is acceptable.
0 The sequence number is not acceptable.

References TCP_SEQ_LEQ, and TCP_SEQ_LT.

Referenced by TcpInput().

```
VOID TcpTrimInWnd ( IN TCP_CB *     Tcb,
                    IN NET_BUF *    Nbuf
                  )
```

Trim off the data outside the tcb's receive window.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.
[in] *Nbuf* Pointer to the NET_BUF containing the received tcp segment.

References TcpTrimSegment().

Referenced by TcpInput().

```
VOID TcpTrimSegment ( IN NET_BUF *     Nbuf,
                      IN TCP_SEQNO     Left,
                      IN TCP_SEQNO     Right
                    )
```

Trim the data; SYN and FIN to fit into the window defined by Left and Right.

Parameters:

[in] *Nbuf* The buffer that contains a received TCP segment without an IP header.
[in] *Left* The sequence number of the window's left edge.
[in] *Right* The sequence number of the window's right edge.

References _TCP_SEG::End, _TCP_SEG::Flag, _TCP_SEG::Seq, TCP_CLEAR_FLG, TCP_FLG_FIN, TCP_FLG_ON, TCP_FLG_SYN, TCP_FLG_URG, TCP_SEQ_GT, TCP_SEQ_LEQ, TCP_SEQ_LT, TCP_SUB_SEQ, TCPSEG_NETBUF, TcpVerifySegment(), and _TCP_SEG::Urg.

Referenced by TcpAdjustSndQue(), TcpQueueData(), and TcpTrimInWnd().

Generated on Mon Sep 28 08:49:56 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/TcpDxe/TcpIo.c File Reference

Functions

VOID EFIAPI TcpRxCallback (IN EFI_STATUS Status, IN UINT8 IcmpErr, IN EFI_NET_SESSION_DATA *NetSession, IN NET_BUF *Pkt, IN VOID *Context)

INTN TcpSendIpPacket (IN TCP_CB *Tcb, IN NET_BUF *Nbuf, IN EFI_IP_ADDRESS *Src, IN EFI_IP_ADDRESS *Dest, IN UINT8 Version)

EFI_STATUS Tcp6RefreshNeighbor (IN TCP_CB *Tcb, IN EFI_IP_ADDRESS *Neighbor, IN UINT32 Timeout)

Detailed Description

Implementation of I/O interfaces between TCP and IpIoLib.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS Tcp6RefreshNeighbor ( IN TCP_CB *            Tcb,
                                 IN EFI_IP_ADDRESS *    Neighbor,
                                 IN UINT32              Timeout
                               )
```

Refresh the remote peer's Neighbor Cache State if already exists.

Parameters:

[in]	*Tcb*	Pointer to the TCP_CB of this TCP instance.
[in]	*Neighbor*	Source address of the TCP segment.
[in]	*Timeout*	Time in 100-ns units that this entry will remain in the neighbor cache. A value of zero means that the entry is permanent. A value of non-zero means that the entry is dynamic and will be deleted after Timeout.

Return values:

EFI_SUCCESS	Successfully updated the neighbor relationship.
EFI_NOT_STARTED	The IpIo is not configured.
EFI_INVALID_PARAMETER	Any input parameter is invalid.
EFI_OUT_OF_RESOURCES	Failed to allocate some resources.
EFI_NOT_FOUND	This entry is not in the neighbor table.

References _TCP_SERVICE_DATA::IpIo, _TCP_SOCKET::ProtoReserved, and _TCP_PROTO_DATA::TcpService.

Referenced by TcpInput().

```
VOID EFIAPI TcpRxCallback ( IN EFI_STATUS                 Status,
                            IN UINT8                      IcmpErr,
                            IN EFI_NET_SESSION_DATA *     NetSession,
                            IN NET_BUF *                  Pkt,
                            IN VOID *                     Context
                          )
```

Packet receive callback function provided to IP_IO, used to call the proper function to handle the packet received by IP.

Parameters:

[in]	*Status*	Result of the receive request.
[in]	*IcmpErr*	Valid when Status is EFI_ICMP_ERROR.
[in]	*NetSession*	The IP session for the received packet.
[in]	*Pkt*	Packet received.
[in]	*Context*	The data provided by the user for the received packet when the callback is registered in IP_IO_OPEN_DATA::RcvdContext. This is an optional parameter that may be NULL.

References TcpIcmpInput(), and TcpInput().

Referenced by TcpCreateService().

```
INTN TcpSendIpPacket ( IN TCP_CB *            Tcb,
                       IN NET_BUF *           Nbuf,
                       IN EFI_IP_ADDRESS *    Src,
                       IN EFI_IP_ADDRESS *    Dest,
                       IN UINT8               Version
                     )
```

Send the segment to IP via IpIo function.

Parameters:

[in]	*Tcb*	Pointer to the TCP_CB of this TCP instance.
[in]	*Nbuf*	Pointer to the TCP segment to be sent.
[in]	*Src*	Source address of the TCP segment.
[in]	*Dest*	Destination address of the TCP segment.
[in]	*Version*	IP_VERSION_4 or IP_VERSION_6

Return values:

0 The segment was sent out successfully.

-1 The segment failed to send.

References _TCP_SERVICE_DATA::IpIo, _TCP_SOCKET::ProtoReserved, and _TCP_PROTO_DATA::TcpService.

Referenced by TcpResetConnection(), TcpSendReset(), and TcpTransmitSegment().

Generated on Mon Sep 28 08:49:56 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/TcpDxe/TcpMain.c File Reference

Functions

EFI_STATUS TcpChkDataBuf (IN UINT32 DataLen, IN UINT32 FragmentCount, IN EFI_TCP4_FRAGMENT_DATA *FragmentTable)
EFI_STATUS EFIAPI Tcp4GetModeData (IN EFI_TCP4_PROTOCOL *This, OUT EFI_TCP4_CONNECTION_STATE *Tcp4State, OUT EFI_TCP4_CONFIG_DATA *Tcp4ConfigData, OUT EFI_IP4_MODE_DATA *Ip4ModeData, OUT EFI_MANAGED_NETWORK_CONFIG_DATA *MnpConfigData, OUT EFI_SIMPLE_NETWORK_MODE *SnpModeData)
EFI_STATUS EFIAPI Tcp4Configure (IN EFI_TCP4_PROTOCOL *This, IN EFI_TCP4_CONFIG_DATA *TcpConfigData)
EFI_STATUS EFIAPI Tcp4Routes (IN EFI_TCP4_PROTOCOL *This, IN BOOLEAN DeleteRoute, IN EFI_IPv4_ADDRESS *SubnetAddress, IN EFI_IPv4_ADDRESS *SubnetMask, IN EFI_IPv4_ADDRESS *GatewayAddress)
EFI_STATUS EFIAPI Tcp4Connect (IN EFI_TCP4_PROTOCOL *This, IN EFI_TCP4_CONNECTION_TOKEN *ConnectionToken)
EFI_STATUS EFIAPI Tcp4Accept (IN EFI_TCP4_PROTOCOL *This, IN EFI_TCP4_LISTEN_TOKEN *ListenToken)
EFI_STATUS EFIAPI Tcp4Transmit (IN EFI_TCP4_PROTOCOL *This, IN EFI_TCP4_IO_TOKEN *Token)
EFI_STATUS EFIAPI Tcp4Receive (IN EFI_TCP4_PROTOCOL *This, IN EFI_TCP4_IO_TOKEN *Token)
EFI_STATUS EFIAPI Tcp4Close (IN EFI_TCP4_PROTOCOL *This, IN EFI_TCP4_CLOSE_TOKEN *CloseToken)
EFI_STATUS EFIAPI Tcp4Cancel (IN EFI_TCP4_PROTOCOL *This, IN EFI_TCP4_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Tcp4Poll (IN EFI_TCP4_PROTOCOL *This)
EFI_STATUS EFIAPI Tcp6GetModeData (IN EFI_TCP6_PROTOCOL *This, OUT EFI_TCP6_CONNECTION_STATE *Tcp6State, OUT EFI_TCP6_CONFIG_DATA *Tcp6ConfigData, OUT EFI_IP6_MODE_DATA *Ip6ModeData, OUT EFI_MANAGED_NETWORK_CONFIG_DATA *MnpConfigData, OUT EFI_SIMPLE_NETWORK_MODE *SnpModeData)
EFI_STATUS EFIAPI Tcp6Configure (IN EFI_TCP6_PROTOCOL *This, IN EFI_TCP6_CONFIG_DATA *Tcp6ConfigData)
EFI_STATUS EFIAPI Tcp6Connect (IN EFI_TCP6_PROTOCOL *This, IN EFI_TCP6_CONNECTION_TOKEN *ConnectionToken)
EFI_STATUS EFIAPI Tcp6Accept (IN EFI_TCP6_PROTOCOL *This, IN EFI_TCP6_LISTEN_TOKEN *ListenToken)
EFI_STATUS EFIAPI Tcp6Transmit (IN EFI_TCP6_PROTOCOL *This, IN EFI_TCP6_IO_TOKEN *Token)
EFI_STATUS EFIAPI Tcp6Receive (IN EFI_TCP6_PROTOCOL *This, IN EFI_TCP6_IO_TOKEN *Token)
EFI_STATUS EFIAPI Tcp6Close (IN EFI_TCP6_PROTOCOL *This, IN EFI_TCP6_CLOSE_TOKEN *CloseToken)
EFI_STATUS EFIAPI Tcp6Cancel (IN EFI_TCP6_PROTOCOL *This, IN EFI_TCP6_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Tcp6Poll (IN EFI_TCP6_PROTOCOL *This)

Detailed Description

Implementation of EFI_TCP4_PROTOCOL and EFI_TCP6_PROTOCOL.

(C) Copyright 2014 Hewlett-Packard Development Company, L.P.
Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI Tcp4Accept ( IN EFI_TCP4_PROTOCOL *        This,
                               IN EFI_TCP4_LISTEN_TOKEN *    ListenToken
                             )
```

Listen on the passive instance to accept an incoming connection request.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*ListenToken*	Pointer to the listen token to return when operation finishes.

Return values:

EFI_SUCCESS	The listen token was queued successfully.
EFI_NOT_STARTED	The EFI_TCP4_PROTOCOL instance hasn't been configured.
EFI_ACCESS_DENIED	The instatnce is not a passive one or it is not in Tcp4StateListen state or a same listen token has already existed in the listen token queue of this TCP instance.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_OUT_OF_RESOURCES	Could not allocate enough resources to finish the operation.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References SOCK_FROM_THIS, and SockAccept().

```
EFI_STATUS EFIAPI Tcp4Cancel ( IN EFI_TCP4_PROTOCOL *                 This,
                               IN EFI_TCP4_COMPLETION_TOKEN *         Token
                             )
```

Abort an asynchronous connection, listen, transmission or receive request.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*Token*	Pointer to a token that has been issued by Connect(), Accept(), Transmit() or Receive(). If NULL, all pending tokens issued by the four functions listed above will be aborted.

Return values:

EFI_UNSUPPORTED The operation is not supported in the current implementation.

```
EFI_STATUS EFIAPI Tcp4Close ( IN EFI_TCP4_PROTOCOL *        This,
                              IN EFI_TCP4_CLOSE_TOKEN *     CloseToken
                            )
```

Disconnecting a TCP connection gracefully or reset a TCP connection.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*CloseToken*	Pointer to the close token to return when operation finishes.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_STARTED	The EFI_TCP4_PROTOCOL instance hasn't been configured.
EFI_ACCESS_DENIED	One or more of the following are TRUE: * Configure() has been called with TcpConfigData set to NULL, and this function has not returned. Previous Close() call on this instance has not finished.
EFI_INVALID_PARAMETER	One ore more parameters are invalid.
EFI_OUT_OF_RESOURCES	Could not allocate enough resources to finish the operation.
EFI_DEVICE_ERROR	Any unexpected category error not belonging to those listed above.

References SOCK_FROM_THIS, and SockClose().

```
EFI_STATUS EFIAPI Tcp4Configure ( IN EFI_TCP4_PROTOCOL *        This,
                                  IN EFI_TCP4_CONFIG_DATA *     TcpConfigData
                                )
```

Initialize or brutally reset the operational parameters for this EFI TCPv4 instance.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*TcpConfigData*	Pointer to the configure data to configure the instance. Optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The operational settings were set, changed, or reset successfully.
EFI_NO_MAPPING	When using a default address, configuration (through DHCP, BOOTP, RARP, etc.) is not finished.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_ACCESS_DENIED	Configuring TCP instance when it is already configured.
EFI_DEVICE_ERROR	An unexpected network or system error occurred.
EFI_UNSUPPORTED	One or more of the control options are not supported in the implementation.
EFI_OUT_OF_RESOURCES	Could not allocate enough system resources.

References _TCP_SOCKET::ConfigureState, SO_NO_MAPPING, SOCK_FROM_THIS, SockConfigure(), and SockFlush().

```
EFI_STATUS EFIAPI Tcp4Connect ( IN EFI_TCP4_PROTOCOL *                This,
```

IN EFI_TCP4_CONNECTION_TOKEN * *ConnectionToken*
)

Initiate a non-blocking TCP connection request for an active TCP instance.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*ConnectionToken*	Pointer to the connection token to return when the TCP three way handshake finishes.

Return values:

EFI_SUCCESS	The connection request successfully initiated.
EFI_NOT_STARTED	This EFI_TCP4_PROTOCOL instance hasn't been configured.
EFI_ACCESS_DENIED	The instance is not configured as an active one, or it is not in Tcp4StateClosed state.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_OUT_OF_RESOURCES	The driver can't allocate enough resources to initiate the active open.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References SOCK_FROM_THIS, and SockConnect().

EFI_STATUS EFIAPI Tcp4GetModeData
(IN EFI_TCP4_PROTOCOL * *This*,
OUT EFI_TCP4_CONNECTION_STATE * *Tcp4State*,
OUT EFI_TCP4_CONFIG_DATA * *Tcp4ConfigData*,
OUT EFI_IP4_MODE_DATA * *Ip4ModeData*,
OUT EFI_MANAGED_NETWORK_CONFIG_DATA * *MnpConfigData*,
OUT EFI_SIMPLE_NETWORK_MODE * *SnpModeData*
)

Get the current operational status.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[out]	*Tcp4State*	Pointer to the buffer to receive the current TCP state. Optional parameter that may be NULL.
[out]	*Tcp4ConfigData*	Pointer to the buffer to receive the current TCP configuration. Optional parameter that may be NULL.
[out]	*Ip4ModeData*	Pointer to the buffer to receive the current IPv4 configuration. Optional parameter that may be NULL.
[out]	*MnpConfigData*	Pointer to the buffer to receive the current MNP configuration data indirectly used by the TCPv4 Instance. Optional parameter that may be NULL.
[out]	*SnpModeData*	Pointer to the buffer to receive the current SNP configuration data indirectly used by the TCPv4 Instance. Optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The mode data was read.
EFI_NOT_STARTED	No configuration data is available because this instance hasn't been started.

EFI_INVALID_PARAMETER This is NULL.

References _TCP4_MODE_DATA::Ip4ModeData, _TCP4_MODE_DATA::MnpConfigData, _TCP4_MODE_DATA::SnpModeData, SOCK_FROM_THIS, SockGetMode(), _TCP4_MODE_DATA::Tcp4ConfigData, and _TCP4_MODE_DATA::Tcp4State.

```
EFI_STATUS EFIAPI Tcp4Poll ( IN EFI_TCP4_PROTOCOL * This )
```

Poll to receive incoming data and transmit outgoing segments.

Parameters:

[in] *This* Pointer to the EFI_TCP4_PROTOCOL instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_INVALID_PARAMETER	This is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_NOT_READY	No incoming or outgoing data was processed.
EFI_TIMEOUT	Data was dropped out of the transmission or receive queue. Consider increasing the polling rate.

References _TCP_SOCKET::ProtoHandler, SOCK_FROM_THIS, and SOCK_POLL.

```
EFI_STATUS EFIAPI Tcp4Receive ( IN EFI_TCP4_PROTOCOL * This,
                                IN EFI_TCP4_IO_TOKEN * Token
                              )
```

Place an asynchronous receive request into the receiving queue.

Parameters:

[in] *This* Pointer to the EFI_TCP4_PROTOCOL instance.

[in] *Token* Pointer to a token that is associated with the receive data descriptor.

Return values:

EFI_SUCCESS	The receive completion token was cached
EFI_NOT_STARTED	The EFI_TCP4_PROTOCOL instance hasn't been configured.
EFI_NO_MAPPING	When using a default address, configuration (DHCP, BOOTP, RARP, etc.) is not finished yet.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_OUT_OF_RESOURCES	The receive completion token could not be queued due to a lack of system resources.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_ACCESS_DENIED	One or more of the following conditions is TRUE: A receive completion token with the same Token->CompletionToken.Event was already in the receive queue. * The current instance is in Tcp4StateClosed state. * The current instance is a passive one and it is in Tcp4StateListen state. * User has called Close() to disconnect this

	connection.
EFI_CONNECTION_FIN	The communication peer has closed the connection, and there is no any buffered data in the receive buffer of this instance.
EFI_NOT_READY	The receive request could not be queued because the receive queue is full.

References SOCK_FROM_THIS, SockRcv(), and TcpChkDataBuf().

```
EFI_STATUS EFIAPI Tcp4Routes ( IN EFI_TCP4_PROTOCOL *  This,
                               IN BOOLEAN              DeleteRoute,
                               IN EFI_IPv4_ADDRESS *   SubnetAddress,
                               IN EFI_IPv4_ADDRESS *   SubnetMask,
                               IN EFI_IPv4_ADDRESS *   GatewayAddress
                             )
```

Add or delete routing entries.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*DeleteRoute*	If TRUE, delete the specified route from routing table; if FALSE, add the specified route to routing table.
[in]	*SubnetAddress*	The destination network.
[in]	*SubnetMask*	The subnet mask for the destination network.
[in]	*GatewayAddress*	The gateway address for this route.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_STARTED	The EFI_TCP4_PROTOCOL instance has not been configured.
EFI_NO_MAPPING	When using a default address, configuration (through DHCP, BOOTP, RARP, etc.) is not finished.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_OUT_OF_RESOURCES	Could not allocate enough resources to add the entry to the routing table.
EFI_NOT_FOUND	This route is not in the routing table.
EFI_ACCESS_DENIED	This route is already in the routing table.
EFI_UNSUPPORTED	The TCP driver does not support this operation.

References _TCP4_ROUTE_INFO::DeleteRoute, _TCP4_ROUTE_INFO::GatewayAddress, SOCK_FROM_THIS, SockRoute(), _TCP4_ROUTE_INFO::SubnetAddress, and _TCP4_ROUTE_INFO::SubnetMask.

```
EFI_STATUS EFIAPI Tcp4Transmit ( IN EFI_TCP4_PROTOCOL *  This,
                                 IN EFI_TCP4_IO_TOKEN *  Token
                               )
```

Queues outgoing data into the transmit queue

Parameters:

[in] *This* Pointer to the EFI_TCP4_PROTOCOL instance.
[in] *Token* Pointer to the completion token to queue to the transmit queue.

Return values:

EFI_SUCCESS	The data has been queued for transmission.
EFI_NOT_STARTED	The EFI_TCP4_PROTOCOL instance hasn't been configured.
EFI_NO_MAPPING	When using a default address, configuration (DHCP, BOOTP, RARP, etc.) is not finished yet.
EFI_INVALID_PARAMETER	One or more parameters are invalid
EFI_ACCESS_DENIED	One or more of the following conditions is TRUE: A transmit completion token with the same Token->CompletionToken.Event was already in the transmission queue. * The current instance is in Tcp4StateClosed state. * The current instance is a passive one and it is in Tcp4StateListen state. * User has called Close() to disconnect this connection.
EFI_NOT_READY	The completion token could not be queued because the transmit queue is full.
EFI_OUT_OF_RESOURCES	Could not queue the transmit data because of a resource shortage.
EFI_NETWORK_UNREACHABLE	There is no route to the destination network or address.

References SOCK_FROM_THIS, SockSend(), and TcpChkDataBuf().

```
EFI_STATUS EFIAPI Tcp6Accept ( IN EFI_TCP6_PROTOCOL *        This,
                               IN EFI_TCP6_LISTEN_TOKEN *    ListenToken
                             )
```

Listen on the passive instance to accept an incoming connection request. This is a nonblocking operation.

The Accept() function initiates an asynchronous accept request to wait for an incoming connection on the passive TCP instance. If a remote peer successfully establishes a connection with this instance, a new TCP instance will be created and its handle will be returned in ListenToken->NewChildHandle. The newly created instance is configured by inheriting the passive instance's configuration and is ready for use upon return. The new instance is in the Tcp6StateEstablished state.

The ListenToken->CompletionToken.Event will be signaled when a new connection is accepted, when a user aborts the listen or when a connection is reset.

This function only can be called when a current TCP instance is in Tcp6StateListen state.

Parameters:

[in] *This* Pointer to the EFI_TCP6_PROTOCOL instance.
[in] *ListenToken* Pointer to the listen token to return when operation finishes.

Return values:

EFI_SUCCESS	The listen token queued successfully.
EFI_NOT_STARTED	This EFI TCPv6 Protocol instance has not been configured.
EFI_ACCESS_DENIED	One or more of the following are TRUE: ◊ This instance is not a passive instance. ◊ This instance is not in Tcp6StateListen state. ◊ The same listen token has already existed in the listen token queue of this TCP instance.
EFI_INVALID_PARAMETER	One or more of the following are TRUE:

◊ This is NULL.
◊ ListenToken is NULL.
◊ ListentToken->CompletionToken.Event is NULL.

EFI_OUT_OF_RESOURCES Could not allocate enough resource to finish the operation.
EFI_DEVICE_ERROR Any unexpected error not belonging to a category listed above.

References SOCK_FROM_THIS, and SockAccept().

```
EFI_STATUS EFIAPI Tcp6Cancel ( IN EFI_TCP6_PROTOCOL *              This,
                               IN EFI_TCP6_COMPLETION_TOKEN *      Token
                             )
```

Abort an asynchronous connection, listen, transmission, or receive request.

The Cancel() function aborts a pending connection, listen, transmit, or receive request.

If Token is not NULL and the token is in the connection, listen, transmission, or receive queue when it is being cancelled, its Token->Status will be set to EFI_ABORTED, and then Token->Event will be signaled.

If the token is not in one of the queues, which usually means that the asynchronous operation has completed, EFI_NOT_FOUND is returned.

If Token is NULL all asynchronous token issued by Connect(), Accept(), Transmit(), and Receive() will be aborted.

Parameters:

[in] *This* Pointer to the EFI_TCP6_PROTOCOL instance.
[in] *Token* Pointer to a token that has been issued by EFI_TCP6_PROTOCOL.Connect(), EFI_TCP6_PROTOCOL.Accept(), EFI_TCP6_PROTOCOL.Transmit() or EFI_TCP6_PROTOCOL.Receive(). If NULL, all pending tokens issued by above four functions will be aborted. Type EFI_TCP6_COMPLETION_TOKEN is defined in EFI_TCP_PROTOCOL.Connect().

Return values:

EFI_UNSUPPORTED The implementation does not support this function.

```
EFI_STATUS EFIAPI Tcp6Close ( IN EFI_TCP6_PROTOCOL *        This,
                              IN EFI_TCP6_CLOSE_TOKEN *     CloseToken
                            )
```

Disconnecting a TCP connection gracefully or reset a TCP connection. This function is a nonblocking operation.

Initiate an asynchronous close token to the TCP driver. After Close() is called, any buffered transmission data will be sent by the TCP driver, and the current instance will have a graceful close working flow described as RFC 793 if AbortOnClose is set to FALSE. Otherwise, a rest packet will be sent by TCP driver to fast disconnect this connection. When the close operation completes successfully the TCP instance is in Tcp6StateClosed state, all pending asynchronous operations are signaled, and any buffers used for TCP network traffic are flushed.

Parameters:

[in] *This* Pointer to the EFI_TCP6_PROTOCOL instance.
[in] *CloseToken* Pointer to the close token to return when operation finishes.

Return values:

EFI_SUCCESS	The Close() was called successfully.
EFI_NOT_STARTED	This EFI TCPv6 Protocol instance has not been configured.
EFI_ACCESS_DENIED	One or more of the following are TRUE: ◊ CloseToken or CloseToken->CompletionToken.Event is already in use. ◊ Previous Close() call on this instance has not finished.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: ◊ This is NULL. ◊ CloseToken is NULL. ◊ CloseToken->CompletionToken.Event is NULL.
EFI_OUT_OF_RESOURCES	Could not allocate enough resource to finish the operation.
EFI_DEVICE_ERROR	Any unexpected error not belonging to error categories given above.

References SOCK_FROM_THIS, and SockClose().

```
EFI_STATUS EFIAPI Tcp6Configure ( IN EFI_TCP6_PROTOCOL *        This,
                                  IN EFI_TCP6_CONFIG_DATA *     Tcp6ConfigData
                                )
```

Initialize or brutally reset the operational parameters for this EFI TCPv6 instance.

The Configure() function does the following:

- Initialize this TCP instance, i.e., initialize the communication end settings and specify active open or passive open for an instance.
- Reset this TCP instance brutally, i.e., cancel all pending asynchronous tokens, flush transmission and receiving buffer directly without informing the communication peer.

No other TCPv6 Protocol operation except Poll() can be executed by this instance until it is configured properly. For an active TCP instance, after a proper configuration it may call Connect() to initiate a three-way handshake. For a passive TCP instance, its state transits to Tcp6StateListen after configuration, and Accept() may be called to listen the incoming TCP connection requests. If Tcp6ConfigData is set to NULL, the instance is reset. The resetting process will be done brutally, the state machine will be set to Tcp6StateClosed directly, the receive queue and transmit queue will be flushed, and no traffic is allowed through this instance.

Parameters:

`[in]` *This*	Pointer to the EFI_TCP6_PROTOCOL instance.
`[in]` *Tcp6ConfigData*	Pointer to the configure data to configure the instance. If Tcp6ConfigData is set to NULL, the instance is reset.

Return values:

EFI_SUCCESS	The operational settings were set, changed, or reset successfully.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_INVALID_PARAMETER	One or more of the following conditions are TRUE: ◊ This is NULL. ◊ Tcp6ConfigData->AccessPoint.StationAddress is neither zero nor one of the configured IP addresses in the underlying IPv6 driver. ◊ Tcp6ConfigData->AccessPoint.RemoteAddress isn't a valid unicast IPv6 address.

◊ Tcp6ConfigData->AccessPoint.RemoteAddress is zero or Tcp6ConfigData->AccessPoint.RemotePort is zero when Tcp6ConfigData->AccessPoint.ActiveFlag is TRUE.
◊ A same access point has been configured in other TCP instance properly.

EFI_ACCESS_DENIED	Configuring a TCP instance when it is configured without calling Configure() with NULL to reset it.
EFI_UNSUPPORTED	One or more of the control options are not supported in the implementation.
EFI_OUT_OF_RESOURCES	Could not allocate enough system resources when executing Configure().
EFI_DEVICE_ERROR	An unexpected network or system error occurred.

References _TCP_SOCKET::ConfigureState, SO_NO_MAPPING, SOCK_FROM_THIS, SockConfigure(), and SockFlush().

```
EFI_STATUS EFIAPI Tcp6Connect ( IN EFI_TCP6_PROTOCOL *              This,
                                IN EFI_TCP6_CONNECTION_TOKEN *      ConnectionToken
                              )
```

Initiate a nonblocking TCP connection request for an active TCP instance.

The Connect() function will initiate an active open to the remote peer configured in a current TCP instance if it is configured active. If the connection succeeds or fails due to any error, the ConnectionToken->CompletionToken.Event will be signaled and ConnectionToken->CompletionToken.Status will be updated accordingly. This function can only be called for the TCP instance in the Tcp6StateClosed state. The instance will transfer into Tcp6StateSynSent if the function returns EFI_SUCCESS. If a TCP three-way handshake succeeds, its state will become Tcp6StateEstablished. Otherwise, the state will return to Tcp6StateClosed.

Parameters:

`[in]`	*This*	Pointer to the EFI_TCP6_PROTOCOL instance.
`[in]`	*ConnectionToken*	Pointer to the connection token to return when the TCP three way handshake finishes.

Return values:

EFI_SUCCESS	The connection request successfully initiated and the state of this TCP instance has been changed to Tcp6StateSynSent.
EFI_NOT_STARTED	This EFI TCPv6 Protocol instance has not been configured.
EFI_ACCESS_DENIED	One or more of the following conditions are TRUE: ◊ This instance is not configured as an active one. ◊ This instance is not in Tcp6StateClosed state.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: ◊ This is NULL. ◊ ConnectionToken is NULL. ◊ ConnectionToken->CompletionToken.Event is NULL.
EFI_OUT_OF_RESOURCES	The driver can't allocate enough resources to initiate the active open.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References SOCK_FROM_THIS, and SockConnect().

```
                              ( IN EFI_TCP6_PROTOCOL *                  This,
```

```
EFI_STATUS EFIAPI
Tcp6GetModeData
                                OUT EFI_TCP6_CONNECTION_STATE *          Tcp6State,
                                OUT EFI_TCP6_CONFIG_DATA *               Tcp6ConfigData,
                                OUT EFI_IP6_MODE_DATA *                  Ip6ModeData,
                                OUT
                                EFI_MANAGED_NETWORK_CONFIG_DATA *        MnpConfigData,
                                OUT EFI_SIMPLE_NETWORK_MODE *            SnpModeData
                              )
```

Get the current operational status.

The GetModeData() function copies the current operational settings of this EFI TCPv6 Protocol instance into user-supplied buffers. This function can also be used to retrieve the operational setting of underlying drivers such as IPv6, MNP, or SNP.

Parameters:

[in]	*This*	Pointer to the EFI_TCP6_PROTOCOL instance.
[out]	*Tcp6State*	The buffer in which the current TCP state is returned. Optional parameter that may be NULL.
[out]	*Tcp6ConfigData*	The buffer in which the current TCP configuration is returned. Optional parameter that may be NULL.
[out]	*Ip6ModeData*	The buffer in which the current IPv6 configuration data used by the TCP instance is returned. Optional parameter that may be NULL.
[out]	*MnpConfigData*	The buffer in which the current MNP configuration data indirectly used by the TCP instance is returned. Optional parameter that may be NULL.
[out]	*SnpModeData*	The buffer in which the current SNP mode data indirectly used by the TCP instance is returned. Optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The mode data was read.
EFI_NOT_STARTED	No configuration data is available because this instance hasn't been started.
EFI_INVALID_PARAMETER	This is NULL.

References _TCP6_MODE_DATA::Ip6ModeData, _TCP6_MODE_DATA::MnpConfigData, _TCP6_MODE_DATA::SnpModeData, SOCK_FROM_THIS, SockGetMode(), _TCP6_MODE_DATA::Tcp6ConfigData, and _TCP6_MODE_DATA::Tcp6State.

```
                                IN
EFI_STATUS EFIAPI Tcp6Poll  (   EFI_TCP6_PROTOCOL   This   )
                                *
```

Poll to receive incoming data and transmit outgoing segments.

The Poll() function increases the rate that data is moved between the network and application, and can be called when the TCP instance is created successfully. Its use is optional.

Parameters:

[in] *This* Pointer to the EFI_TCP6_PROTOCOL instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.

EFI_INVALID_PARAMETER	This is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_NOT_READY	No incoming or outgoing data is processed.
EFI_TIMEOUT	Data was dropped out of the transmission or receive queue. Consider increasing the polling rate.

References _TCP_SOCKET::ProtoHandler, SOCK_FROM_THIS, and SOCK_POLL.

```
EFI_STATUS EFIAPI Tcp6Receive ( IN EFI_TCP6_PROTOCOL * This,
                                IN EFI_TCP6_IO_TOKEN * Token
                              )
```

Places an asynchronous receive request into the receiving queue.

The Receive() function places a completion token into the receive packet queue. This function is always asynchronous. The caller must allocate the Token->CompletionToken.Event and the FragmentBuffer used to receive data. The caller also must fill the DataLength that represents the whole length of all FragmentBuffer. When the receive operation completes, the EFI TCPv6 Protocol driver updates the Token->CompletionToken.Status and Token->Packet.RxData fields, and the Token->CompletionToken.Event is signaled. If data obtained, the data and its length will be copied into the FragmentTable; at the same time the full length of received data will be recorded in the DataLength fields. Providing a proper notification function and context for the event enables the user to receive the notification and receiving status. That notification function is guaranteed to not be re-entered.

Parameters:

`[in]`	*This*	Pointer to the EFI_TCP6_PROTOCOL instance.
`[in]`	*Token*	Pointer to a token that is associated with the receive data descriptor.

Return values:

EFI_SUCCESS	The receive completion token was cached.
EFI_NOT_STARTED	This EFI TCPv6 Protocol instance has not been configured.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token->CompletionToken.Event is NULL. ◊ Token->Packet.RxData is NULL. ◊ Token->Packet.RxData->DataLength is 0. ◊ The Token->Packet.RxData->DataLength is not the sum of all FragmentBuffer length in FragmentTable.
EFI_OUT_OF_RESOURCES	The receive completion token could not be queued due to a lack of system resources (usually memory).
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI TCPv6 Protocol instance has been reset to startup defaults.
EFI_ACCESS_DENIED	One or more of the following conditions is TRUE: ◊ A receive completion token with the same Token->CompletionToken.Event was already in the receive queue. ◊ The current instance is in Tcp6StateClosed state.

◊ The current instance is a passive one and it is in Tcp6StateListen state.
◊ User has called Close() to disconnect this connection.

EFI_CONNECTION_FIN	The communication peer has closed the connection and there is no buffered data in the receive buffer of this instance.
EFI_NOT_READY	The receive request could not be queued because the receive queue is full.

References SOCK_FROM_THIS, SockRcv(), and TcpChkDataBuf().

```
EFI_STATUS EFIAPI Tcp6Transmit ( IN EFI_TCP6_PROTOCOL * This,
                                 IN EFI_TCP6_IO_TOKEN * Token
                               )
```

Queues outgoing data into the transmit queue.

The Transmit() function queues a sending request to this TCP instance along with the user data. The status of the token is updated and the event in the token will be signaled once the data is sent out or an error occurs.

Parameters:

[in]	*This*	Pointer to the EFI_TCP6_PROTOCOL instance.
[in]	*Token*	Pointer to the completion token to queue to the transmit queue.

Return values:

EFI_SUCCESS	The data has been queued for transmission.
EFI_NOT_STARTED	This EFI TCPv6 Protocol instance has not been configured.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token->CompletionToken.Event is NULL. ◊ Token->Packet.TxData is NULL. ◊ Token->Packet.FragmentCount is zero. ◊ Token->Packet.DataLength is not equal to the sum of fragment lengths.
EFI_ACCESS_DENIED	One or more of the following conditions are TRUE: ◊ A transmit completion token with the same Token->CompletionToken.Event was already in the transmission queue. ◊ The current instance is in Tcp6StateClosed state. ◊ The current instance is a passive one and it is in Tcp6StateListen state. ◊ User has called Close() to disconnect this connection.
EFI_NOT_READY	The completion token could not be queued because the transmit queue is full.
EFI_OUT_OF_RESOURCES	Could not queue the transmit data because of resource shortage.
EFI_NETWORK_UNREACHABLE	There is no route to the destination network or address.

References SOCK_FROM_THIS, SockSend(), and TcpChkDataBuf().

```
EFI_STATUS TcpChkDataBuf ( IN UINT32                    DataLen,
                           IN UINT32                    FragmentCount,
                           IN EFI_TCP4_FRAGMENT_DATA *  FragmentTable
                         )
```

Check the integrity of the data buffer.

Parameters:

[in]	*DataLen*	The total length of the data buffer.
[in]	*FragmentCount*	The fragment count of the fragment table.
[in]	*FragmentTable*	Pointer to the fragment table of the data buffer.

Return values:

EFI_SUCCESS	The integrity check passed.
EFI_INVALID_PARAMETER	The integrity check failed.

Referenced by Tcp4Receive(), Tcp4Transmit(), Tcp6Receive(), and Tcp6Transmit().

Generated on Mon Sep 28 08:49:56 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/TcpDxe/TcpMain.h File Reference

Data Structures

union TCP_CONFIG_DATA
union TCP_ACCESS_POINT
struct _TCP4_MODE_DATA
struct _TCP6_MODE_DATA
struct _TCP4_ROUTE_INFO
struct TCP_DESTROY_CHILD_IN_HANDLE_

Defines

"memItemRight" valign="bottom">TCP6_KEEP_NEIGHBOR_TIME 30
#define TCP6_REFRESH_NEIGHBOR_TICK 2
#define TCP_EXPIRE_TIME 65535
#define TCP_BASE_ISS 0x4d7e980b
#define TCP_ISS_INCREMENT_1 2048
#define TCP_ISS_INCREMENT_2 100

Typedefs

typedef struct _TCP4_MODE_DATA TCP4_MODE_DATA
typedef struct _TCP6_MODE_DATA TCP6_MODE_DATA
typedef struct _TCP4_ROUTE_INFO TCP4_ROUTE_INFO

Functions

EFI_STATUS EFIAPI Tcp4GetModeData (IN EFI_TCP4_PROT
OUT EFI_TCP4_CONNECTION_STATE
OUT EFI_TCP4_CONFIG_DATA *Tcp4
EFI_IP4_MODE_DATA *Ip4ModeData,
EFI_MANAGED_NETWORK_CONFIG_
*MnpConfigData, OUT EFI_SIMPLE_NE
*SnpModeData)
EFI_STATUS EFIAPI Tcp4Configure (IN EFI_TCP4_PROTOCO
EFI_TCP4_CONFIG_DATA *TcpConfig
EFI_STATUS EFIAPI Tcp4Routes (IN EFI_TCP4_PROTOCOL
BOOLEAN DeleteRoute, IN EFI_IPv4_A
*SubnetAddress, IN EFI_IPv4_ADDRESS
IN EFI_IPv4_ADDRESS *GatewayAddre
EFI_STATUS EFIAPI Tcp4Connect (IN EFI_TCP4_PROTOCOL
EFI_TCP4_CONNECTION_TOKEN *Co
EFI_STATUS EFIAPI Tcp4Accept (IN EFI_TCP4_PROTOCOL
EFI_TCP4_LISTEN_TOKEN *ListenTok
EFI_STATUS EFIAPI Tcp4Transmit (IN EFI_TCP4_PROTOCO
EFI_TCP4_IO_TOKEN *Token)
EFI_STATUS EFIAPI Tcp4Receive (IN EFI_TCP4_PROTOCOL
EFI_TCP4_IO_TOKEN *Token)
EFI_STATUS EFIAPI Tcp4Close (IN EFI_TCP4_PROTOCOL *

EFI_TCP4_CLOSE_TOKEN *CloseToken)
EFI_STATUS EFIAPI Tcp4Cancel (IN EFI_TCP4_PROTOCOL *This, I
EFI_TCP4_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Tcp4Poll (IN EFI_TCP4_PROTOCOL *This)
EFI_STATUS EFIAPI Tcp6GetModeData (IN EFI_TCP6_PROTOCOL *
OUT EFI_TCP6_CONNECTION_STATE *Tcp6S
OUT EFI_TCP6_CONFIG_DATA *Tcp6ConfigD
EFI_IP6_MODE_DATA *Ip6ModeData, OUT
EFI_MANAGED_NETWORK_CONFIG_DATA
*MnpConfigData, OUT EFI_SIMPLE_NETWORI
*SnpModeData)
EFI_STATUS EFIAPI Tcp6Configure (IN EFI_TCP6_PROTOCOL *This
EFI_TCP6_CONFIG_DATA *Tcp6ConfigData)
EFI_STATUS EFIAPI Tcp6Connect (IN EFI_TCP6_PROTOCOL *This,
EFI_TCP6_CONNECTION_TOKEN *Connection
EFI_STATUS EFIAPI Tcp6Accept (IN EFI_TCP6_PROTOCOL *This, I
EFI_TCP6_LISTEN_TOKEN *ListenToken)
EFI_STATUS EFIAPI Tcp6Transmit (IN EFI_TCP6_PROTOCOL *This,
EFI_TCP6_IO_TOKEN *Token)
EFI_STATUS EFIAPI Tcp6Receive (IN EFI_TCP6_PROTOCOL *This,
EFI_TCP6_IO_TOKEN *Token)
EFI_STATUS EFIAPI Tcp6Close (IN EFI_TCP6_PROTOCOL *This, IN
EFI_TCP6_CLOSE_TOKEN *CloseToken)
EFI_STATUS EFIAPI Tcp6Cancel (IN EFI_TCP6_PROTOCOL *This, I
EFI_TCP6_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Tcp6Poll (IN EFI_TCP6_PROTOCOL *This)

Variables

UINT16 mTcp4RandomPort
UINT16 mTcp6RandomPort
CHAR16 * mTcpStateName []
EFI_COMPONENT_NAME_PROTOCOL gTcpComponentName
EFI_COMPONENT_NAME2_PROTOCOL gTcpComponentName2
EFI_UNICODE_STRING_TABLE * gTcpControllerNameTable
LIST_ENTRY mTcpRunQue
LIST_ENTRY mTcpListenQue
TCP_SEQNO mTcpGlobalIss
UINT32 mTcpTick

Detailed Description

Declaration of protocol interfaces in EFI_TCP4_PROTOCOL and EFI_TCP6_PROTOCOL. It is the common head file for all Tcp*.c in TCP driver.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

30 seconds.

Referenced by TcpInput().

#define TCP6_REFRESH_NEIGHBOR_TICK 25

5 seconds, since 1 tick equals 200ms.

Referenced by TcpConfigurePcb(), and TcpInput().

#define TCP_BASE_ISS 0x4d7e980b

The implementation selects the initial send sequence number and the unit to be added when it is increased.

#define TCP_EXPIRE_TIME 65535

Referenced by TcpUpdateTimer().

#define TCP_ISS_INCREMENT_1 2048

Referenced by TcpGetIss().

#define TCP_ISS_INCREMENT_2 100

Referenced by TcpTickingDpc().

Typedef Documentation

typedef struct _TCP4_MODE_DATA TCP4_MODE_DATA

typedef struct _TCP4_ROUTE_INFO TCP4_ROUTE_INFO

typedef struct _TCP6_MODE_DATA TCP6_MODE_DATA

Function Documentation

EFI_STATUS EFIAPI Tcp4Accept (IN EFI_TCP4_PROTOCOL * *This*,

```
                                    IN EFI_TCP4_LISTEN_TOKEN *   ListenToken
                                  )
```

Listen on the passive instance to accept an incoming connection request.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*ListenToken*	Pointer to the listen token to return when operation finishes.

Return values:

EFI_SUCCESS	The listen token has been queued successfully.
EFI_NOT_STARTED	The EFI_TCP4_PROTOCOL instance hasn't been configured.
EFI_ACCESS_DENIED	The instatnce is not a passive one or it is not in Tcp4StateListen state, or a same listen token has already existed in the listen token queue of this TCP instance.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_OUT_OF_RESOURCES	Could not allocate enough resources to finish the operation.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

Listen on the passive instance to accept an incoming connection request.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*ListenToken*	Pointer to the listen token to return when operation finishes.

Return values:

EFI_SUCCESS	The listen token was queued successfully.
EFI_NOT_STARTED	The EFI_TCP4_PROTOCOL instance hasn't been configured.
EFI_ACCESS_DENIED	The instatnce is not a passive one or it is not in Tcp4StateListen state or a same listen token has already existed in the listen token queue of this TCP instance.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_OUT_OF_RESOURCES	Could not allocate enough resources to finish the operation.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References SOCK_FROM_THIS, and SockAccept().

```
EFI_STATUS EFIAPI Tcp4Cancel  ( IN EFI_TCP4_PROTOCOL *                 This,
                                IN EFI_TCP4_COMPLETION_TOKEN *         Token
                                )
```

Abort an asynchronous connection, listen, transmission or receive request.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*Token*	Pointer to a token that has been issued by Connect(), Accept(), Transmit() or Receive(). If NULL, all pending tokens issued by the above four functions will be aborted.

Return values:

EFI_UNSUPPORTED	The operation is not supported in the current implementation.

Abort an asynchronous connection, listen, transmission or receive request.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*Token*	Pointer to a token that has been issued by Connect(), Accept(), Transmit() or Receive(). If NULL, all pending tokens issued by the four functions listed above will be aborted.

Return values:

EFI_UNSUPPORTED	The operation is not supported in the current implementation.

```
EFI_STATUS EFIAPI Tcp4Close ( IN EFI_TCP4_PROTOCOL *        This,
                              IN EFI_TCP4_CLOSE_TOKEN *     CloseToken
                            )
```

Disconnecting a TCP connection gracefully or reset a TCP connection.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*CloseToken*	Pointer to the close token to return when operation finishes.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_STARTED	The EFI_TCP4_PROTOCOL instance hasn't been configured.
EFI_ACCESS_DENIED	One or more of the following are TRUE: * Configure() has been called with TcpConfigData set to NULL and this function has not returned. Previous Close() call on this instance has not finished.
EFI_INVALID_PARAMETER	One ore more parameters are invalid.
EFI_OUT_OF_RESOURCES	Could not allocate enough resources to finish the operation.
EFI_DEVICE_ERROR	Any unexpected error not belonging to the error categories given above.

Disconnecting a TCP connection gracefully or reset a TCP connection.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*CloseToken*	Pointer to the close token to return when operation finishes.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_STARTED	The EFI_TCP4_PROTOCOL instance hasn't been configured.
EFI_ACCESS_DENIED	One or more of the following are TRUE: * Configure() has been called with TcpConfigData set to NULL, and this function has not returned. Previous Close() call on this instance has not finished.
EFI_INVALID_PARAMETER	One ore more parameters are invalid.
EFI_OUT_OF_RESOURCES	Could not allocate enough resources to finish the operation.
EFI_DEVICE_ERROR	Any unexpected category error not belonging to those listed above.

References SOCK_FROM_THIS, and SockClose().

```
EFI_STATUS EFIAPI Tcp4Configure ( IN EFI_TCP4_PROTOCOL *        This,
                                  IN EFI_TCP4_CONFIG_DATA *     TcpConfigData
                                )
```

Initialize or brutally reset the operational parameters for this EFI TCPv4 instance.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*TcpConfigData*	Pointer to the configure data to configure the instance. Optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The operational settings are set, changed, or reset successfully.
EFI_NO_MAPPING	When using a default address, configuration (through DHCP, BOOTP, RARP, etc.) is not finished.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_ACCESS_DENIED	Configuring the TCP instance when it is already configured.
EFI_DEVICE_ERROR	An unexpected network or system error occurred.
EFI_UNSUPPORTED	One or more of the control options are not supported in the implementation.
EFI_OUT_OF_RESOURCES	Could not allocate enough system resources.

Initialize or brutally reset the operational parameters for this EFI TCPv4 instance.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*TcpConfigData*	Pointer to the configure data to configure the instance. Optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The operational settings were set, changed, or reset successfully.
EFI_NO_MAPPING	When using a default address, configuration (through DHCP, BOOTP, RARP, etc.) is not finished.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_ACCESS_DENIED	Configuring TCP instance when it is already configured.
EFI_DEVICE_ERROR	An unexpected network or system error occurred.
EFI_UNSUPPORTED	One or more of the control options are not supported in the implementation.
EFI_OUT_OF_RESOURCES	Could not allocate enough system resources.

References _TCP_SOCKET::ConfigureState, SO_NO_MAPPING, SOCK_FROM_THIS, SockConfigure(), and SockFlush().

```
EFI_STATUS EFIAPI Tcp4Connect ( IN EFI_TCP4_PROTOCOL *              This,
                                IN EFI_TCP4_CONNECTION_TOKEN *      ConnectionToken
                              )
```

Initiate a nonblocking TCP connection request for an active TCP instance.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*ConnectionToken*	Pointer to the connection token to return when the TCP three way handshake finishes.

Return values:

EFI_SUCCESS	The connection request is successfully initiated.

EFI_NOT_STARTED	This EFI_TCP4_PROTOCOL instance hasn't been configured.
EFI_ACCESS_DENIED	The instance is not configured as an active one or it is not in Tcp4StateClosed state.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_OUT_OF_RESOURCES	The driver can't allocate enough resources to initiate the active open.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

Initiate a non-blocking TCP connection request for an active TCP instance.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[in]	*ConnectionToken*	Pointer to the connection token to return when the TCP three way handshake finishes.

Return values:

EFI_SUCCESS	The connection request successfully initiated.
EFI_NOT_STARTED	This EFI_TCP4_PROTOCOL instance hasn't been configured.
EFI_ACCESS_DENIED	The instance is not configured as an active one, or it is not in Tcp4StateClosed state.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_OUT_OF_RESOURCES	The driver can't allocate enough resources to initiate the active open.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References SOCK_FROM_THIS, and SockConnect().

```
EFI_STATUS EFIAPI
Tcp4GetModeData
    ( IN EFI_TCP4_PROTOCOL *                    This,
      OUT EFI_TCP4_CONNECTION_STATE *           Tcp4State,
      OUT EFI_TCP4_CONFIG_DATA *                Tcp4ConfigData,
      OUT EFI_IP4_MODE_DATA *                   Ip4ModeData,
      OUT
      EFI_MANAGED_NETWORK_CONFIG_DATA *         MnpConfigData,
      OUT EFI_SIMPLE_NETWORK_MODE *             SnpModeData
    )
```

Get the current operational status.

Parameters:

[in]	*This*	Pointer to the EFI_TCP4_PROTOCOL instance.
[out]	*Tcp4State*	Pointer to the buffer to receive the current TCP state. Optional parameter that may be NULL.
[out]	*Tcp4ConfigData*	Pointer to the buffer to receive the current TCP configuration. Optional parameter that may be NULL.
[out]	*Ip4ModeData*	Pointer to the buffer to receive the current IPv4 configuration. Optional parameter that may be NULL.
[out]	*MnpConfigData*	Pointer to the buffer to receive the current MNP configuration data indirectly used by the TCPv4 Instance. Optional parameter that may be NULL.
[out]	*SnpModeData*	Pointer to the buffer to receive the current SNP configuration data indirectly used by the TCPv4 Instance. Optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The mode data was read.
EFI_NOT_STARTED	No configuration data is available because this instance hasn't been started.
EFI_INVALID_PARAMETER	This is NULL.

References _TCP4_MODE_DATA::Ip4ModeData, _TCP4_MODE_DATA::MnpConfigData, _TCP4_MODE_DATA::SnpModeData, SOCK_FROM_THIS, SockGetMode(), _TCP4_MODE_DATA::Tcp4ConfigData, and _TCP4_MODE_DATA::Tcp4State.

```
EFI_STATUS EFIAPI Tcp4Poll ( IN EFI_TCP4_PROTOCOL * This )
```

Poll to receive incoming data and transmit outgoing segments.

Parameters:

`[in]` *This* Pointer to the EFI_TCP4_PROTOCOL instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_INVALID_PARAMETER	This is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_NOT_READY	No incoming or outgoing data was processed.
EFI_TIMEOUT	Data was dropped out of the transmission or receive queue. Consider increasing the polling rate.

References _TCP_SOCKET::ProtoHandler, SOCK_FROM_THIS, and SOCK_POLL.

```
EFI_STATUS EFIAPI Tcp4Receive ( IN EFI_TCP4_PROTOCOL * This,
                                IN EFI_TCP4_IO_TOKEN * Token )
```

Place an asynchronous receive request into the receiving queue.

Parameters:

`[in]` *This* Pointer to the EFI_TCP4_PROTOCOL instance.

`[in]` *Token* Pointer to a token that is associated with the receive data descriptor.

Return values:

EFI_SUCCESS	The receive completion token was cached.
EFI_NOT_STARTED	The EFI_TCP4_PROTOCOL instance hasn't been configured.
EFI_NO_MAPPING	When using a default address, configuration (DHCP, BOOTP, RARP, etc.) is not finished yet.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_OUT_OF_RESOURCES	The receive completion token could not be queued due to a lack of system resources.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_ACCESS_DENIED	

	One or more of the following conditions is TRUE: A receive completion token with the same Token->CompletionToken.Event was already in the receive queue. * The current instance is in Tcp4StateClosed state. * The current instance is a passive one and it is in Tcp4StateListen state. * User has called Close() to disconnect this connection.
EFI_CONNECTION_FIN	The communication peer has closed the connection and there is no buffered data in the receive buffer of this instance.
EFI_NOT_READY	The receive request could not be queued because the receive queue is full.

Place an asynchronous receive request into the receiving queue.

Parameters:

`[in]` *This* Pointer to the EFI_TCP4_PROTOCOL instance.

`[in]` *Token* Pointer to a token that is associated with the receive data descriptor.

Return values:

EFI_SUCCESS	The receive completion token was cached
EFI_NOT_STARTED	The EFI_TCP4_PROTOCOL instance hasn't been configured.
EFI_NO_MAPPING	When using a default address, configuration (DHCP, BOOTP, RARP, etc.) is not finished yet.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_OUT_OF_RESOURCES	The receive completion token could not be queued due to a lack of system resources.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_ACCESS_DENIED	One or more of the following conditions is TRUE: A receive completion token with the same Token->CompletionToken.Event was already in the receive queue. * The current instance is in Tcp4StateClosed state. * The current instance is a passive one and it is in Tcp4StateListen state. * User has called Close() to disconnect this connection.
EFI_CONNECTION_FIN	The communication peer has closed the connection, and there is no any buffered data in the receive buffer of this instance.
EFI_NOT_READY	The receive request could not be queued because the receive queue is full.

References SOCK_FROM_THIS, SockRcv(), and TcpChkDataBuf().

```
EFI_STATUS EFIAPI Tcp4Routes ( IN EFI_TCP4_PROTOCOL *   This,
                               IN BOOLEAN               DeleteRoute,
                               IN EFI_IPv4_ADDRESS *    SubnetAddress,
                               IN EFI_IPv4_ADDRESS *    SubnetMask,
                               IN EFI_IPv4_ADDRESS *    GatewayAddress
                             )
```

Add or delete routing entries.

Parameters:

`[in]` *This* Pointer to the EFI_TCP4_PROTOCOL instance.

`[in]` *DeleteRoute* If TRUE, delete the specified route from routing table; if FALSE, add the specified route to routing table.

[in] *SubnetAddress* The destination network.
[in] *SubnetMask* The subnet mask for the destination network.
[in] *GatewayAddress* The gateway address for this route.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_STARTED	The EFI_TCP4_PROTOCOL instance has not been configured.
EFI_NO_MAPPING	When using a default address, configuration (through DHCP, BOOTP, RARP, etc.) is not finished.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_OUT_OF_RESOURCES	Could not allocate enough resources to add the entry to the routing table.
EFI_NOT_FOUND	This route is not in the routing table.
EFI_ACCESS_DENIED	This route is already in the routing table.
EFI_UNSUPPORTED	The TCP driver does not support this operation.

References _TCP4_ROUTE_INFO::DeleteRoute, _TCP4_ROUTE_INFO::GatewayAddress, SOCK_FROM_THIS, SockRoute(), _TCP4_ROUTE_INFO::SubnetAddress, and _TCP4_ROUTE_INFO::SubnetMask.

```
EFI_STATUS EFIAPI Tcp4Transmit ( IN EFI_TCP4_PROTOCOL * This,
                                 IN EFI_TCP4_IO_TOKEN * Token
                               )
```

Queues outgoing data into the transmit queue

Parameters:

[in] *This* Pointer to the EFI_TCP4_PROTOCOL instance
[in] *Token* Pointer to the completion token to queue to the transmit queue

Return values:

EFI_SUCCESS	The data has been queued for transmission
EFI_NOT_STARTED	The EFI_TCP4_PROTOCOL instance hasn't been configured.
EFI_NO_MAPPING	When using a default address, configuration (DHCP, BOOTP, RARP, etc.) is not finished yet.
EFI_INVALID_PARAMETER	One or more parameters are invalid
EFI_ACCESS_DENIED	One or more of the following conditions is TRUE: A transmit completion token with the same Token-> CompletionToken.Event was already in the transmission queue. * The current instance is in Tcp4StateClosed state * The current instance is a passive one and it is in Tcp4StateListen state. * User has called Close() to disconnect this connection.
EFI_NOT_READY	The completion token could not be queued because the transmit queue is full.
EFI_OUT_OF_RESOURCES	Could not queue the transmit data because of a resource shortage.
EFI_NETWORK_UNREACHABLE	There is no route to the destination network or address.

Queues outgoing data into the transmit queue

Parameters:

[in] *This* Pointer to the EFI_TCP4_PROTOCOL instance.

[in] *Token* Pointer to the completion token to queue to the transmit queue.

Return values:

EFI_SUCCESS	The data has been queued for transmission.
EFI_NOT_STARTED	The EFI_TCP4_PROTOCOL instance hasn't been configured.
EFI_NO_MAPPING	When using a default address, configuration (DHCP, BOOTP, RARP, etc.) is not finished yet.
EFI_INVALID_PARAMETER	One or more parameters are invalid
EFI_ACCESS_DENIED	One or more of the following conditions is TRUE: A transmit completion token with the same Token-> CompletionToken.Event was already in the transmission queue. * The current instance is in Tcp4StateClosed state. * The current instance is a passive one and it is in Tcp4StateListen state. * User has called Close() to disconnect this connection.
EFI_NOT_READY	The completion token could not be queued because the transmit queue is full.
EFI_OUT_OF_RESOURCES	Could not queue the transmit data because of a resource shortage.
EFI_NETWORK_UNREACHABLE	There is no route to the destination network or address.

References SOCK_FROM_THIS, SockSend(), and TcpChkDataBuf().

```
EFI_STATUS EFIAPI Tcp6Accept ( IN EFI_TCP6_PROTOCOL *        This,
                               IN EFI_TCP6_LISTEN_TOKEN *    ListenToken
                             )
```

Listen on the passive instance to accept an incoming connection request. This is a nonblocking operation.

The Accept() function initiates an asynchronous accept request to wait for an incoming connection on the passive TCP instance. If a remote peer successfully establishes a connection with this instance, a new TCP instance will be created and its handle will be returned in ListenToken->NewChildHandle. The newly created instance is configured by inheriting the passive instance's configuration, and is ready for use upon return. The new instance is in the Tcp6StateEstablished state.

The ListenToken->CompletionToken.Event will be signaled when a new connection is accepted, user aborts the listen or connection is reset.

This function only can be called when the current TCP instance is in Tcp6StateListen state.

Parameters:

[in] *This* Pointer to the EFI_TCP6_PROTOCOL instance.

[in] *ListenToken* Pointer to the listen token to return when the operation finishes.

Return values:

EFI_SUCCESS	The listen token was been queued successfully.
EFI_NOT_STARTED	This EFI TCPv6 Protocol instance has not been configured.
EFI_ACCESS_DENIED	One or more of the following are TRUE: ◊ This instance is not a passive instance. ◊ This instance is not in Tcp6StateListen state. ◊ The same listen token has already existed in the listen token queue of this TCP instance.

EFI_INVALID_PARAMETER	One or more of the following are TRUE: ◊ This is NULL. ◊ ListenToken is NULL. ◊ ListentToken->CompletionToken.Event is NULL.
EFI_OUT_OF_RESOURCES	Could not allocate enough resources to finish the operation.
EFI_DEVICE_ERROR	Any unexpected error not belonging to the error categories given above.

Listen on the passive instance to accept an incoming connection request. This is a nonblocking operation.

The Accept() function initiates an asynchronous accept request to wait for an incoming connection on the passive TCP instance. If a remote peer successfully establishes a connection with this instance, a new TCP instance will be created and its handle will be returned in ListenToken->NewChildHandle. The newly created instance is configured by inheriting the passive instance's configuration and is ready for use upon return. The new instance is in the Tcp6StateEstablished state.

The ListenToken->CompletionToken.Event will be signaled when a new connection is accepted, when a user aborts the listen or when a connection is reset.

This function only can be called when a current TCP instance is in Tcp6StateListen state.

Parameters:

`[in]`	*This*	Pointer to the EFI_TCP6_PROTOCOL instance.
`[in]`	*ListenToken*	Pointer to the listen token to return when operation finishes.

Return values:

EFI_SUCCESS	The listen token queued successfully.
EFI_NOT_STARTED	This EFI TCPv6 Protocol instance has not been configured.
EFI_ACCESS_DENIED	One or more of the following are TRUE: ◊ This instance is not a passive instance. ◊ This instance is not in Tcp6StateListen state. ◊ The same listen token has already existed in the listen token queue of this TCP instance.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: ◊ This is NULL. ◊ ListenToken is NULL. ◊ ListentToken->CompletionToken.Event is NULL.
EFI_OUT_OF_RESOURCES	Could not allocate enough resource to finish the operation.
EFI_DEVICE_ERROR	Any unexpected error not belonging to a category listed above.

References SOCK_FROM_THIS, and SockAccept().

```
EFI_STATUS EFIAPI Tcp6Cancel ( IN EFI_TCP6_PROTOCOL *              This,
                               IN EFI_TCP6_COMPLETION_TOKEN *      Token
                             )
```

Abort an asynchronous connection, listen, transmission or receive request.

The Cancel() function aborts a pending connection, listen, transmit or receive request.

If Token is not NULL and the token is in the connection, listen, transmission or receive queue when it is being cancelled, its Token->Status will be set to EFI_ABORTED and then Token->Event will be signaled.

If the token is not in one of the queues, which usually means that the asynchronous operation has completed, EFI_NOT_FOUND is returned.

If Token is NULL all asynchronous token issued by Connect(), Accept(), Transmit() and Receive() will be aborted.

Parameters:

`[in]` *This* Pointer to the EFI_TCP6_PROTOCOL instance.

`[in]` *Token* Pointer to a token that has been issued by EFI_TCP6_PROTOCOL.Connect(), EFI_TCP6_PROTOCOL.Accept(), EFI_TCP6_PROTOCOL.Transmit() or EFI_TCP6_PROTOCOL.Receive(). If NULL, all pending tokens issued by above four functions will be aborted. Type EFI_TCP6_COMPLETION_TOKEN is defined in EFI_TCP_PROTOCOL.Connect().

Return values:

EFI_UNSUPPORTED The implementation does not support this function.

Abort an asynchronous connection, listen, transmission, or receive request.

The Cancel() function aborts a pending connection, listen, transmit, or receive request.

If Token is not NULL and the token is in the connection, listen, transmission, or receive queue when it is being cancelled, its Token->Status will be set to EFI_ABORTED, and then Token->Event will be signaled.

If the token is not in one of the queues, which usually means that the asynchronous operation has completed, EFI_NOT_FOUND is returned.

If Token is NULL all asynchronous token issued by Connect(), Accept(), Transmit(), and Receive() will be aborted.

Parameters:

`[in]` *This* Pointer to the EFI_TCP6_PROTOCOL instance.

`[in]` *Token* Pointer to a token that has been issued by EFI_TCP6_PROTOCOL.Connect(), EFI_TCP6_PROTOCOL.Accept(), EFI_TCP6_PROTOCOL.Transmit() or EFI_TCP6_PROTOCOL.Receive(). If NULL, all pending tokens issued by above four functions will be aborted. Type EFI_TCP6_COMPLETION_TOKEN is defined in EFI_TCP_PROTOCOL.Connect().

Return values:

EFI_UNSUPPORTED The implementation does not support this function.

```
EFI_STATUS EFIAPI Tcp6Close ( IN EFI_TCP6_PROTOCOL *        This,
                              IN EFI_TCP6_CLOSE_TOKEN *     CloseToken
                            )
```

Disconnecting a TCP connection gracefully or reset a TCP connection. This function is a nonblocking operation.

Initiate an asynchronous close token to the TCP driver. After Close() is called, any buffered transmission data will be sent by the TCP driver, and the current instance will have a graceful close working flow described as RFC 793 if AbortOnClose is set to FALSE, otherwise, a rest packet will be sent by TCP driver to fast disconnect this connection. When the close operation completes successfully the TCP instance is in Tcp6StateClosed state, all pending asynchronous operations are signaled, and any buffers used for TCP network traffic are flushed.

Parameters:

[in]	*This*	Pointer to the EFI_TCP6_PROTOCOL instance.
[in]	*CloseToken*	Pointer to the close token to return when operation finishes.

Return values:

EFI_SUCCESS	The Close() was called successfully.
EFI_NOT_STARTED	This EFI TCPv6 Protocol instance has not been configured.
EFI_ACCESS_DENIED	One or more of the following are TRUE: ◊ CloseToken or CloseToken->CompletionToken.Event is already in use. ◊ Previous Close() call on this instance has not finished.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: ◊ This is NULL. ◊ CloseToken is NULL. ◊ CloseToken->CompletionToken.Event is NULL.
EFI_OUT_OF_RESOURCES	Could not allocate enough resources to finish the operation.
EFI_DEVICE_ERROR	Any unexpected error not belonging to the error categories given above.

Disconnecting a TCP connection gracefully or reset a TCP connection. This function is a nonblocking operation.

Initiate an asynchronous close token to the TCP driver. After Close() is called, any buffered transmission data will be sent by the TCP driver, and the current instance will have a graceful close working flow described as RFC 793 if AbortOnClose is set to FALSE. Otherwise, a rest packet will be sent by TCP driver to fast disconnect this connection. When the close operation completes successfully the TCP instance is in Tcp6StateClosed state, all pending asynchronous operations are signaled, and any buffers used for TCP network traffic are flushed.

Parameters:

[in]	*This*	Pointer to the EFI_TCP6_PROTOCOL instance.
[in]	*CloseToken*	Pointer to the close token to return when operation finishes.

Return values:

EFI_SUCCESS	The Close() was called successfully.
EFI_NOT_STARTED	This EFI TCPv6 Protocol instance has not been configured.
EFI_ACCESS_DENIED	One or more of the following are TRUE: ◊ CloseToken or CloseToken->CompletionToken.Event is already in use. ◊ Previous Close() call on this instance has not finished.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: ◊ This is NULL. ◊ CloseToken is NULL. ◊ CloseToken->CompletionToken.Event is NULL.
EFI_OUT_OF_RESOURCES	Could not allocate enough resource to finish the operation.
EFI_DEVICE_ERROR	Any unexpected error not belonging to error categories given above.

References SOCK_FROM_THIS, and SockClose().

```
EFI_STATUS EFIAPI Tcp6Configure ( IN EFI_TCP6_PROTOCOL *        This,
                                  IN EFI_TCP6_CONFIG_DATA *     Tcp6ConfigData
                                )
```

Initialize or brutally reset the operational parameters for this EFI TCPv6 instance.

The Configure() function does the following:

- Initialize this TCP instance, i.e., initialize the communication end settings and specify active open or passive open for an instance.
- Reset this TCP instance brutally, i.e., cancel all pending asynchronous tokens, flush transmission and receiving buffer directly without informing the communication peer.

No other TCPv6 Protocol operation except Poll() can be executed by this instance until it is configured properly. For an active TCP instance, after a proper configuration it may call Connect() to initiates the three-way handshake. For a passive TCP instance, its state will transit to Tcp6StateListen after configuration, and Accept() may be called to listen the incoming TCP connection requests. If Tcp6ConfigData is set to NULL, the instance is reset. Resetting process will be done brutally, the state machine will be set to Tcp6StateClosed directly, the receive queue and transmit queue will be flushed, and no traffic is allowed through this instance.

Parameters:

`[in]`	*This*	Pointer to the EFI_TCP6_PROTOCOL instance.
`[in]`	*Tcp6ConfigData*	Pointer to the configure data to configure the instance. If Tcp6ConfigData is set to NULL, the instance is reset.

Return values:

EFI_SUCCESS	The operational settings were set, changed, or reset successfully.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_INVALID_PARAMETER	One or more of the following conditions are TRUE: ◊ This is NULL. ◊ Tcp6ConfigData->AccessPoint.StationAddress is neither zero nor one of the configured IP addresses in the underlying IPv6 driver. ◊ Tcp6ConfigData->AccessPoint.RemoteAddress isn't a valid unicast IPv6 address. ◊ Tcp6ConfigData->AccessPoint.RemoteAddress is zero or Tcp6ConfigData->AccessPoint.RemotePort is zero when Tcp6ConfigData->AccessPoint.ActiveFlag is TRUE. ◊ A same access point has been configured in other TCP instance properly.
EFI_ACCESS_DENIED	Configuring TCP instance when it is configured without calling Configure() with NULL to reset it.
EFI_UNSUPPORTED	One or more of the control options are not supported in the implementation.
EFI_OUT_OF_RESOURCES	Could not allocate enough system resources when executing Configure().
EFI_DEVICE_ERROR	An unexpected network or system error occurred.

Initialize or brutally reset the operational parameters for this EFI TCPv6 instance.

The Configure() function does the following:

- Initialize this TCP instance, i.e., initialize the communication end settings and specify active open or passive open for an instance.
- Reset this TCP instance brutally, i.e., cancel all pending asynchronous tokens, flush transmission and receiving buffer directly without informing the communication peer.

No other TCPv6 Protocol operation except Poll() can be executed by this instance until it is configured properly. For an active TCP instance, after a proper configuration it may call Connect() to initiate a three-way handshake. For a passive TCP instance, its state transits to Tcp6StateListen after configuration, and Accept() may be called to listen the incoming TCP connection requests. If Tcp6ConfigData is set to NULL, the instance is reset. The resetting process will be done brutally, the state machine will be set to Tcp6StateClosed directly, the receive queue and transmit queue will be flushed, and no traffic is allowed through this instance.

Parameters:

`[in]`	*This*	Pointer to the EFI_TCP6_PROTOCOL instance.
`[in]`	*Tcp6ConfigData*	Pointer to the configure data to configure the instance. If Tcp6ConfigData is set to NULL, the instance is reset.

Return values:

EFI_SUCCESS	The operational settings were set, changed, or reset successfully.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_INVALID_PARAMETER	One or more of the following conditions are TRUE: ◊ This is NULL. ◊ Tcp6ConfigData->AccessPoint.StationAddress is neither zero nor one of the configured IP addresses in the underlying IPv6 driver. ◊ Tcp6ConfigData->AccessPoint.RemoteAddress isn't a valid unicast IPv6 address. ◊ Tcp6ConfigData->AccessPoint.RemoteAddress is zero or Tcp6ConfigData->AccessPoint.RemotePort is zero when Tcp6ConfigData->AccessPoint.ActiveFlag is TRUE. ◊ A same access point has been configured in other TCP instance properly.
EFI_ACCESS_DENIED	Configuring a TCP instance when it is configured without calling Configure() with NULL to reset it.
EFI_UNSUPPORTED	One or more of the control options are not supported in the implementation.
EFI_OUT_OF_RESOURCES	Could not allocate enough system resources when executing Configure().
EFI_DEVICE_ERROR	An unexpected network or system error occurred.

References _TCP_SOCKET::ConfigureState, SO_NO_MAPPING, SOCK_FROM_THIS, SockConfigure(), and SockFlush().

```
EFI_STATUS EFIAPI Tcp6Connect ( IN EFI_TCP6_PROTOCOL *              This,
                                IN EFI_TCP6_CONNECTION_TOKEN *      ConnectionToken
                              )
```

Initiate a nonblocking TCP connection request for an active TCP instance.

The Connect() function will initiate an active open to the remote peer configured in current TCP instance if it is configured active. If the connection succeeds or fails due to an error, the ConnectionToken->CompletionToken.Event will be signaled, and ConnectionToken->CompletionToken.Status will be updated accordingly. This function can only be called for the TCP instance in Tcp6StateClosed state. The instance will transfer into Tcp6StateSynSent if the function returns EFI_SUCCESS. If TCP three-way handshake succeeds, its state will become Tcp6StateEstablished; otherwise, the state will return to Tcp6StateClosed.

Parameters:

[in]	*This*	Pointer to the EFI_TCP6_PROTOCOL instance.
[in]	*ConnectionToken*	Pointer to the connection token to return when the TCP three-way handshake finishes.

Return values:

EFI_SUCCESS	The connection request successfully initiated and the state of this TCP instance has been changed to Tcp6StateSynSent.
EFI_NOT_STARTED	This EFI TCPv6 Protocol instance has not been configured.
EFI_ACCESS_DENIED	One or more of the following conditions are TRUE: ◊ This instance is not configured as an active instance. ◊ This instance is not in Tcp6StateClosed state.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: ◊ This is NULL. ◊ ConnectionToken is NULL. ◊ ConnectionToken->CompletionToken.Event is NULL.
EFI_OUT_OF_RESOURCES	The driver can't allocate enough resources to initiate the active open.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

Initiate a nonblocking TCP connection request for an active TCP instance.

The Connect() function will initiate an active open to the remote peer configured in a current TCP instance if it is configured active. If the connection succeeds or fails due to any error, the ConnectionToken->CompletionToken.Event will be signaled and ConnectionToken->CompletionToken.Status will be updated accordingly. This function can only be called for the TCP instance in the Tcp6StateClosed state. The instance will transfer into Tcp6StateSynSent if the function returns EFI_SUCCESS. If a TCP three-way handshake succeeds, its state will become Tcp6StateEstablished. Otherwise, the state will return to Tcp6StateClosed.

Parameters:

[in]	*This*	Pointer to the EFI_TCP6_PROTOCOL instance.
[in]	*ConnectionToken*	Pointer to the connection token to return when the TCP three way handshake finishes.

Return values:

EFI_SUCCESS	The connection request successfully initiated and the state of this TCP instance has been changed to Tcp6StateSynSent.
EFI_NOT_STARTED	This EFI TCPv6 Protocol instance has not been configured.
EFI_ACCESS_DENIED	One or more of the following conditions are TRUE: ◊ This instance is not configured as an active one. ◊ This instance is not in Tcp6StateClosed state.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: ◊ This is NULL. ◊ ConnectionToken is NULL. ◊ ConnectionToken->CompletionToken.Event is NULL.
EFI_OUT_OF_RESOURCES	The driver can't allocate enough resources to initiate the active open.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References SOCK_FROM_THIS, and SockConnect().

EFI_STATUS EFIAPI Tcp6GetModeData (IN EFI_TCP6_PROTOCOL * *This*,

OUT EFI_TCP6_CONNECTION_STATE *	*Tcp6State*,
OUT EFI_TCP6_CONFIG_DATA *	*Tcp6ConfigData*,
OUT EFI_IP6_MODE_DATA *	*Ip6ModeData*,
OUT EFI_MANAGED_NETWORK_CONFIG_DATA *	*MnpConfigData*,
OUT EFI_SIMPLE_NETWORK_MODE *	*SnpModeData*

)

Get the current operational status.

The GetModeData() function copies the current operational settings of this EFI TCPv6 Protocol instance into user-supplied buffers. This function can also be used to retrieve the operational setting of underlying drivers such as IPv6, MNP, or SNP.

Parameters:

[in]	*This*	Pointer to the EFI_TCP6_PROTOCOL instance.
[out]	*Tcp6State*	The buffer in which the current TCP state is returned. Optional parameter that may be NULL.
[out]	*Tcp6ConfigData*	The buffer in which the current TCP configuration is returned. Optional parameter that may be NULL.
[out]	*Ip6ModeData*	The buffer in which the current IPv6 configuration data used by the TCP instance is returned. Optional parameter that may be NULL.
[out]	*MnpConfigData*	The buffer in which the current MNP configuration data used indirectly by the TCP instance is returned. Optional parameter that may be NULL.
[out]	*SnpModeData*	The buffer in which the current SNP mode data used indirectly by the TCP instance is returned. Optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The mode data was read.
EFI_NOT_STARTED	No configuration data is available because this instance hasn't been started.
EFI_INVALID_PARAMETER	This is NULL.

Get the current operational status.

The GetModeData() function copies the current operational settings of this EFI TCPv6 Protocol instance into user-supplied buffers. This function can also be used to retrieve the operational setting of underlying drivers such as IPv6, MNP, or SNP.

Parameters:

[in]	*This*	Pointer to the EFI_TCP6_PROTOCOL instance.
[out]	*Tcp6State*	The buffer in which the current TCP state is returned. Optional parameter that may be NULL.
[out]	*Tcp6ConfigData*	The buffer in which the current TCP configuration is returned. Optional parameter that may be NULL.
[out]	*Ip6ModeData*	The buffer in which the current IPv6 configuration data used by the TCP instance is returned. Optional parameter that may be NULL.
[out]	*MnpConfigData*	The buffer in which the current MNP configuration data indirectly used by the TCP instance is returned. Optional parameter that may be NULL.
[out]	*SnpModeData*	The buffer in which the current SNP mode data indirectly used by the TCP instance is returned. Optional parameter that may be NULL.

Return values:

EFI_SUCCESS	The mode data was read.
EFI_NOT_STARTED	No configuration data is available because this instance hasn't been started.
EFI_INVALID_PARAMETER	This is NULL.

References _TCP6_MODE_DATA::Ip6ModeData, _TCP6_MODE_DATA::MnpConfigData, _TCP6_MODE_DATA::SnpModeData, SOCK_FROM_THIS, SockGetMode(), _TCP6_MODE_DATA::Tcp6ConfigData, and _TCP6_MODE_DATA::Tcp6State.

```
EFI_STATUS EFIAPI Tcp6Poll ( IN EFI_TCP6_PROTOCOL * This )
```

Poll to receive incoming data and transmit outgoing segments.

The Poll() function increases the rate that data is moved between the network and application and can be called when the TCP instance is created successfully. Its use is optional.

Parameters:

`[in]` *This* Pointer to the EFI_TCP6_PROTOCOL instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_INVALID_PARAMETER	This is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_NOT_READY	No incoming or outgoing data is processed.
EFI_TIMEOUT	Data was dropped out of the transmission or receive queue. Consider increasing the polling rate.

Poll to receive incoming data and transmit outgoing segments.

The Poll() function increases the rate that data is moved between the network and application, and can be called when the TCP instance is created successfully. Its use is optional.

Parameters:

`[in]` *This* Pointer to the EFI_TCP6_PROTOCOL instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_INVALID_PARAMETER	This is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_NOT_READY	No incoming or outgoing data is processed.
EFI_TIMEOUT	Data was dropped out of the transmission or receive queue. Consider increasing the polling rate.

References _TCP_SOCKET::ProtoHandler, SOCK_FROM_THIS, and SOCK_POLL.

```
EFI_STATUS EFIAPI Tcp6Receive ( IN EFI_TCP6_PROTOCOL * This,
                                IN EFI_TCP6_IO_TOKEN * Token
```

)

Places an asynchronous receive request into the receiving queue.

The Receive() function places a completion token into the receive packet queue. This function is always asynchronous. The caller must allocate the Token->CompletionToken.Event and the FragmentBuffer used to receive data. The caller also must fill the DataLength, which represents the whole length of all FragmentBuffer. When the receive operation completes, the EFI TCPv6 Protocol driver updates the Token->CompletionToken.Status and Token->Packet.RxData fields, and the Token->CompletionToken.Event is signaled. If data is obtained, the data and its length will be copied into the FragmentTable. At the same time the full length of received data will be recorded in the DataLength fields. Providing a proper notification function and context for the event enables the user to receive the notification and receiving status. That notification function is guaranteed to not be re-entered.

Parameters:

`[in]` *This* Pointer to the EFI_TCP6_PROTOCOL instance.

`[in]` *Token* Pointer to a token that is associated with the receive data descriptor.

Return values:

EFI_SUCCESS	The receive completion token was cached.
EFI_NOT_STARTED	This EFI TCPv6 Protocol instance has not been configured.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token->CompletionToken.Event is NULL. ◊ Token->Packet.RxData is NULL. ◊ Token->Packet.RxData->DataLength is 0. ◊ The Token->Packet.RxData->DataLength is not the sum of all FragmentBuffer length in FragmentTable.
EFI_OUT_OF_RESOURCES	The receive completion token could not be queued due to a lack of system resources (usually memory).
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI TCPv6 Protocol instance has been reset to startup defaults.
EFI_ACCESS_DENIED	One or more of the following conditions is TRUE: ◊ A receive completion token with the same Token->CompletionToken.Event was already in the receive queue. ◊ The current instance is in Tcp6StateClosed state. ◊ The current instance is a passive one and it is in Tcp6StateListen state. ◊ The user has called Close() to disconnect this connection.
EFI_CONNECTION_FIN	The communication peer has closed the connection, and there is no buffered data in the receive buffer of this instance.
EFI_NOT_READY	The receive request could not be queued because the receive queue is full.

Places an asynchronous receive request into the receiving queue.

The Receive() function places a completion token into the receive packet queue. This function is always asynchronous. The caller must allocate the Token->CompletionToken.Event and the FragmentBuffer used to receive data. The caller also must fill the DataLength that represents the whole length of all FragmentBuffer. When the receive operation completes, the EFI TCPv6 Protocol driver updates the

Token->CompletionToken.Status and Token->Packet.RxData fields, and the Token->CompletionToken.Event is signaled. If data obtained, the data and its length will be copied into the FragmentTable; at the same time the full length of received data will be recorded in the DataLength fields. Providing a proper notification function and context for the event enables the user to receive the notification and receiving status. That notification function is guaranteed to not be re-entered.

Parameters:

[in]	*This*	Pointer to the EFI_TCP6_PROTOCOL instance.
[in]	*Token*	Pointer to a token that is associated with the receive data descriptor.

Return values:

EFI_SUCCESS	The receive completion token was cached.
EFI_NOT_STARTED	This EFI TCPv6 Protocol instance has not been configured.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token->CompletionToken.Event is NULL. ◊ Token->Packet.RxData is NULL. ◊ Token->Packet.RxData->DataLength is 0. ◊ The Token->Packet.RxData->DataLength is not the sum of all FragmentBuffer length in FragmentTable.
EFI_OUT_OF_RESOURCES	The receive completion token could not be queued due to a lack of system resources (usually memory).
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI TCPv6 Protocol instance has been reset to startup defaults.
EFI_ACCESS_DENIED	One or more of the following conditions is TRUE: ◊ A receive completion token with the same Token->CompletionToken.Event was already in the receive queue. ◊ The current instance is in Tcp6StateClosed state. ◊ The current instance is a passive one and it is in Tcp6StateListen state. ◊ User has called Close() to disconnect this connection.
EFI_CONNECTION_FIN	The communication peer has closed the connection and there is no buffered data in the receive buffer of this instance.
EFI_NOT_READY	The receive request could not be queued because the receive queue is full.

References SOCK_FROM_THIS, SockRcv(), and TcpChkDataBuf().

```
EFI_STATUS EFIAPI Tcp6Transmit ( IN EFI_TCP6_PROTOCOL * This,
                                 IN EFI_TCP6_IO_TOKEN * Token
                               )
```

Queues outgoing data into the transmit queue.

The Transmit() function queues a sending request to this TCP instance along with the user data. The status of the token is updated and the event in the token will be signaled once the data is sent out or some error occurs.

Parameters:

[in] *This* Pointer to the EFI_TCP6_PROTOCOL instance.

[in] *Token* Pointer to the completion token to queue to the transmit queue.

Return values:

EFI_SUCCESS	The data has been queued for transmission.
EFI_NOT_STARTED	This EFI TCPv6 Protocol instance has not been configured.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token->CompletionToken.Event is NULL. ◊ Token->Packet.TxData is NULL. ◊ Token->Packet.FragmentCount is zero. ◊ Token->Packet.DataLength is not equal to the sum of fragment lengths.
EFI_ACCESS_DENIED	One or more of the following conditions are TRUE: ◊ A transmit completion token with the same Token->CompletionToken.Event was already in the transmission queue. ◊ The current instance is in Tcp6StateClosed state. ◊ The current instance is a passive one and it is in Tcp6StateListen state. ◊ User has called Close() to disconnect this connection.
EFI_NOT_READY	The completion token could not be queued because the transmit queue is full.
EFI_OUT_OF_RESOURCES	Could not queue the transmit data because of a resource shortage.
EFI_NETWORK_UNREACHABLE	There is no route to the destination network or address.

Queues outgoing data into the transmit queue.

The Transmit() function queues a sending request to this TCP instance along with the user data. The status of the token is updated and the event in the token will be signaled once the data is sent out or an error occurs.

Parameters:

[in] *This* Pointer to the EFI_TCP6_PROTOCOL instance.

[in] *Token* Pointer to the completion token to queue to the transmit queue.

Return values:

EFI_SUCCESS	The data has been queued for transmission.
EFI_NOT_STARTED	This EFI TCPv6 Protocol instance has not been configured.
EFI_NO_MAPPING	The underlying IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: ◊ This is NULL. ◊ Token is NULL. ◊ Token->CompletionToken.Event is NULL. ◊ Token->Packet.TxData is NULL. ◊ Token->Packet.FragmentCount is zero.

	◊ Token->Packet.DataLength is not equal to the sum of fragment lengths.
EFI_ACCESS_DENIED	One or more of the following conditions are TRUE: ◊ A transmit completion token with the same Token->CompletionToken.Event was already in the transmission queue. ◊ The current instance is in Tcp6StateClosed state. ◊ The current instance is a passive one and it is in Tcp6StateListen state. ◊ User has called Close() to disconnect this connection.
EFI_NOT_READY	The completion token could not be queued because the transmit queue is full.
EFI_OUT_OF_RESOURCES	Could not queue the transmit data because of resource shortage.
EFI_NETWORK_UNREACHABLE	There is no route to the destination network or address.

References SOCK_FROM_THIS, SockSend(), and TcpChkDataBuf().

Variable Documentation

EFI_COMPONENT_NAME_PROTOCOL gTcpComponentName
EFI Component Name Protocol

Referenced by TcpComponentNameGetControllerName(), TcpComponentNameGetDriverName(), TcpDriverEntryPoint(), UpdateTcp4Name(), and UpdateTcp6Name().

EFI_COMPONENT_NAME2_PROTOCOL gTcpComponentName2
EFI Component Name 2 Protocol

Referenced by TcpDriverEntryPoint(), UpdateTcp4Name(), and UpdateTcp6Name().

EFI_UNICODE_STRING_TABLE* gTcpControllerNameTable

Referenced by TcpComponentNameGetControllerName(), UpdateTcp4Name(), and UpdateTcp6Name().

UINT16 mTcp4RandomPort

Referenced by TcpBind(), and TcpDriverEntryPoint().

UINT16 mTcp6RandomPort

Referenced by TcpBind(), and TcpDriverEntryPoint().

TCP_SEQNO mTcpGlobalIss

Referenced by TcpDriverEntryPoint(), TcpGetIss(), and TcpTickingDpc().

LIST_ENTRY mTcpListenQue

Referenced by TcpFindTcbByPeer(), TcpInsertTcb(), and TcpLocateListenTcb().

LIST_ENTRY mTcpRunQue

Referenced by TcpFindTcbByPeer(), TcpInsertTcb(), TcpLocateTcb(), and TcpTickingDpc().

CHAR16* mTcpStateName[]

Referenced by TcpSetState(), and TcpToSendData().

UINT32 mTcpTick

Referenced by TcpBuildOption(), TcpInput(), TcpPawsOK(), TcpSetTimer(), TcpSynBuildOption(), TcpTickingDpc(), and TcpUpdateTimer().

NetworkPkg/TcpDxe/TcpMisc.c File Reference

Functions

```
          VOID TcpInitTcbLocal (IN OUT TCP_CB *Tcb)
          VOID TcpInitTcbPeer (IN OUT TCP_CB *Tcb, IN TCP_SEG *Seg, IN TCP_OPTION *Opt)
       BOOLEAN TcpIsIpEqual (IN EFI_IP_ADDRESS *Ip1, IN EFI_IP_ADDRESS *Ip2, IN UINT8 Version)
       BOOLEAN TcpIsIpZero (IN EFI_IP_ADDRESS *Ip, IN UINT8 Version)
        TCP_CB * TcpLocateListenTcb (IN TCP_PEER *Local, IN TCP_PEER *Remote, IN UINT8 Version)
       BOOLEAN TcpFindTcbByPeer (IN EFI_IP_ADDRESS *Addr, IN TCP_PORTNO Port, IN UINT8
                Version)
        TCP_CB * TcpLocateTcb (IN TCP_PORTNO LocalPort, IN EFI_IP_ADDRESS *LocalIp, IN
                TCP_PORTNO RemotePort, IN EFI_IP_ADDRESS *RemoteIp, IN UINT8 Version, IN
                BOOLEAN Syn)
          INTN TcpInsertTcb (IN TCP_CB *Tcb)
        TCP_CB * TcpCloneTcb (IN TCP_CB *Tcb)
     TCP_SEQNO TcpGetIss (VOID)
        UINT16 TcpGetRcvMss (IN SOCKET *Sock)
          VOID TcpSetState (IN TCP_CB *Tcb, IN UINT8 State)
        UINT16 TcpChecksum (IN NET_BUF *Nbuf, IN UINT16 HeadSum)
       TCP_SEG * TcpFormatNetbuf (IN TCP_CB *Tcb, IN OUT NET_BUF *Nbuf)
          VOID TcpOnAppConnect (IN OUT TCP_CB *Tcb)
          VOID TcpOnAppClose (IN OUT TCP_CB *Tcb)
          INTN TcpOnAppSend (IN OUT TCP_CB *Tcb)
          VOID TcpOnAppConsume (IN TCP_CB *Tcb)
          VOID TcpOnAppAbort (IN TCP_CB *Tcb)
          VOID TcpResetConnection (IN TCP_CB *Tcb)
    EFI_STATUS TcpInstallDevicePath (IN SOCKET *Sock)
```

Variables

```
    LIST_ENTRY mTcpRunQue
    LIST_ENTRY mTcpListenQue
     TCP_SEQNO mTcpGlobalIss = TCP_BASE_ISS
     CHAR16 * mTcpStateName []
```

Detailed Description

Misc support routines for TCP driver.

(C) Copyright 2014 Hewlett-Packard Development Company, L.P.
Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
UINT16 TcpChecksum ( IN NET_BUF *   Nbuf,
                     IN UINT16      HeadSum
                   )
```

Compute the TCP segment's checksum.

Parameters:

[in]	*Nbuf*	Pointer to the buffer that contains the TCP segment.
[in]	*HeadSum*	The checksum value of the fixed part of pseudo header.

Returns:

The checksum value.

Referenced by TcpInput(), TcpResetConnection(), TcpSendReset(), and TcpTransmitSegment().

```
TCP_CB* TcpCloneTcb ( IN TCP_CB *   Tcb   )
```

Clone a TCP_CB from Tcb.

Parameters:

[in]	*Tcb*	Pointer to the TCP_CB to be cloned.

Returns:

Pointer to the new cloned TCP_CB; if NULL, error condition occurred.

References _TCP_CONTROL_BLOCK::List, _TCP_SOCKET::ProtoReserved, _TCP_CONTROL_BLOCK::RcvQue, _TCP_CONTROL_BLOCK::Sk, _TCP_CONTROL_BLOCK::SndQue, and SockClone().

Referenced by TcpInput().

```
BOOLEAN TcpFindTcbByPeer ( IN EFI_IP_ADDRESS *   Addr,
                           IN TCP_PORTNO         Port,
                           IN UINT8              Version
                         )
```

Try to find one Tcb whose <Ip, Port> equals to <IN Addr, IN Port>.

Parameters:

[in]	*Addr*	Pointer to the IP address needs to match.
[in]	*Port*	The port number needs to match.
[in]	*Version*	IP_VERSION_4 indicates TCP is running on IP4 stack, IP_VERSION_6 indicates TCP is running on IP6 stack.

Return values:

TRUE	The Tcb which matches the <Addr port>=""> pair exists.
FALSE	Otherwise

References _TCP_PEER::Ip, _TCP_SOCKET::IpVersion, _TCP_CONTROL_BLOCK::LocalEnd, mTcpListenQue, mTcpRunQue, _TCP_PEER::Port, _TCP_CONTROL_BLOCK::Sk, and TcpIsIpEqual().

Referenced by TcpBind().

TCP_SEG* TcpFormatNetbuf (IN TCP_CB * *Tcb*,
IN OUT NET_BUF * *Nbuf*
)

Translate the information from the head of the received TCP segment Nbuf contents and fill it into a TCP_SEG structure.

Parameters:

[in]	*Tcb*	Pointer to the TCP_CB of this TCP instance.
[in,out]	*Nbuf*	Pointer to the buffer contains the TCP segment.

Returns:

Pointer to the TCP_SEG that contains the translated TCP head information.

References _TCP_SEG::Ack, _TCP_SEG::End, _TCP_SEG::Flag, _TCP_SEG::Seq, TCP_FLG_FIN, TCP_FLG_ON, TCP_FLG_SYN, TCPSEG_NETBUF, _TCP_SEG::Urg, and _TCP_SEG::Wnd.

Referenced by TcpInput().

TCP_SEQNO TcpGetIss (VOID)

Compute an ISS to be used by a new connection.

Returns:

The resulting ISS.

References mTcpGlobalIss, and TCP_ISS_INCREMENT_1.

Referenced by TcpInitTcbLocal().

UINT16 TcpGetRcvMss (IN SOCKET * *Sock*)

Get the local mss.

Parameters:

[in]	*Sock*	Pointer to the socket to get mss.

Returns:

The mss size.

References _TCP_SERVICE_DATA::IpIo, and _TCP_PROTO_DATA::TcpService.

Referenced by TcpConfigurePcb(), and TcpInitTcbPeer().

VOID TcpInitTcbLocal (IN OUT TCP_CB * *Tcb*)

Initialize the Tcb local related members.

Parameters:

[in,out]	*Tcb*	Pointer to the TCP_CB of this TCP instance.

References GET_RCV_BUFFSIZE, and TcpGetIss().

Referenced by TcpInput(), and TcpOnAppConnect().

```
VOID TcpInitTcbPeer ( IN OUT TCP_CB *   Tcb,
                      IN TCP_SEG *      Seg,
                      IN TCP_OPTION *   Opt
                    )
```

Initialize the peer related members.

Parameters:

[in,out]	*Tcb*	Pointer to the TCP_CB of this TCP instance.
[in]	*Seg*	Pointer to the segment that contains the peer's intial info.
[in]	*Opt*	Pointer to the options announced by the peer.

References TCP_CTRL_NO_TS, TCP_CTRL_NO_WS, TCP_CTRL_RCVD_TS, TCP_CTRL_RCVD_WS, TCP_CTRL_SND_TS, TCP_FLG_ACK, TCP_FLG_ON, TCP_FLG_SYN, TCP_OPTION_RCVD_MSS, TCP_OPTION_RCVD_TS, TCP_OPTION_RCVD_WS, TCP_OPTION_TS_ALIGNED_LEN, TCP_SET_FLG, TcpComputeScale(), and TcpGetRcvMss().

Referenced by TcpInput().

```
INTN TcpInsertTcb ( IN TCP_CB *   Tcb   )
```

Insert a Tcb into the proper queue.

Parameters:

[in]	*Tcb*	Pointer to the TCP_CB to be inserted.

Return values:

0	The Tcb was inserted successfully.
-1	Error condition occurred.

References _TCP_CONTROL_BLOCK::LocalEnd, mTcpListenQue, mTcpRunQue, _TCP_CONTROL_BLOCK::RemoteEnd, TCP_CLOSED, TCP_LISTEN, TCP_PEER_EQUAL, TCP_SYN_RCVD, and TCP_SYN_SENT.

Referenced by TcpConfigurePcb(), and TcpInput().

```
EFI_STATUS TcpInstallDevicePath ( IN SOCKET *   Sock   )
```

Install the device path protocol on the TCP instance.

Parameters:

[in]	*Sock*	Pointer to the socket representing the TCP instance.

Return values:

EFI_SUCCESS	The device path protocol was installed.
other	Failed to install the device path protocol.

References _TCP_SERVICE_DATA::ControllerHandle, _TCP_PEER::Ip, _TCP_CONTROL_BLOCK::LocalEnd, _TCP_PEER::Port, _TCP_CONTROL_BLOCK::RemoteEnd,

_TCP_CONTROL_BLOCK::SubnetMask, _TCP_PROTO_DATA::TcpPcb, _TCP_PROTO_DATA::TcpService, and _TCP_CONTROL_BLOCK::UseDefaultAddr.

Referenced by TcpConfigurePcb(), and TcpSetState().

```
BOOLEAN TcpIsIpEqual ( IN EFI_IP_ADDRESS *  Ip1,
                       IN EFI_IP_ADDRESS *  Ip2,
                       IN UINT8             Version
                     )
```

Check whether one IP address equals the other.

Parameters:

- [in] *Ip1* Pointer to IP address to be checked.
- [in] *Ip2* Pointer to IP address to be checked.
- [in] *Version* IP_VERSION_4 indicates the IP address is an IPv4 address, IP_VERSION_6 indicates the IP address is an IPv6 address.

Return values:

- *TRUE* Ip1 equals Ip2.
- *FALSE* Ip1 does not equal Ip2.

Referenced by TcpFindTcbByPeer().

```
BOOLEAN TcpIsIpZero ( IN EFI_IP_ADDRESS *  Ip,
                      IN UINT8             Version
                    )
```

Check whether one IP address is filled with ZERO.

Parameters:

- [in] *Ip* Pointer to the IP address to be checked.
- [in] *Version* IP_VERSION_4 indicates the IP address is an IPv4 address, IP_VERSION_6 indicates the IP address is an IPv6 address.

Return values:

- *TRUE* Ip is all zero address.
- *FALSE* Ip is not all zero address.

Referenced by TcpLocateListenTcb().

```
TCP_CB* TcpLocateListenTcb ( IN TCP_PEER *  Local,
                             IN TCP_PEER *  Remote,
                             IN UINT8       Version
                           )
```

Locate a listen TCB that matchs the Local and Remote.

Parameters:

[in]	*Local*	Pointer to the local (IP, Port).
[in]	*Remote*	Pointer to the remote (IP, Port).
[in]	*Version*	IP_VERSION_4 indicates TCP is running on IP4 stack, IP_VERSION_6 indicates TCP is running on IP6 stack.

Returns:

Pointer to the TCP_CB with the least number of wildcards, if NULL no match is found.

References _TCP_PEER::Ip, _TCP_SOCKET::IpVersion, _TCP_CONTROL_BLOCK::LocalEnd, mTcpListenQue, _TCP_PEER::Port, _TCP_CONTROL_BLOCK::RemoteEnd, _TCP_CONTROL_BLOCK::Sk, TCP_PEER_MATCH, and TcpIsIpZero().

Referenced by TcpLocateTcb().

```
TCP_CB* TcpLocateTcb ( IN TCP_PORTNO          LocalPort,
                       IN EFI_IP_ADDRESS *    LocalIp,
                       IN TCP_PORTNO          RemotePort,
                       IN EFI_IP_ADDRESS *    RemoteIp,
                       IN UINT8               Version,
                       IN BOOLEAN             Syn
                     )
```

Locate the TCP_CB related to the socket pair.

Parameters:

[in]	*LocalPort*	The local port number.
[in]	*LocalIp*	The local IP address.
[in]	*RemotePort*	The remote port number.
[in]	*RemoteIp*	The remote IP address.
[in]	*Version*	IP_VERSION_4 indicates TCP is running on IP4 stack, IP_VERSION_6 indicates TCP is running on IP6 stack.
[in]	*Syn*	If TRUE, the listen sockets are searched.

Returns:

Pointer to the related TCP_CB. If NULL, no match is found.

References _TCP_PEER::Ip, _TCP_SOCKET::IpVersion, _TCP_CONTROL_BLOCK::List, _TCP_CONTROL_BLOCK::LocalEnd, mTcpRunQue, _TCP_PEER::Port, _TCP_CONTROL_BLOCK::RemoteEnd, _TCP_CONTROL_BLOCK::Sk, TCP_PEER_EQUAL, and TcpLocateListenTcb().

Referenced by TcpIcmpInput(), and TcpInput().

```
VOID TcpOnAppAbort ( IN TCP_CB *   Tcb   )
```

Abort the connection by sending a reset segment. Called when the application wants to abort the connection.

Parameters:

[in]	*Tcb*	Pointer to the TCP_CB of the TCP instance.

References TCP_CLOSE_WAIT, TCP_CLOSED, TCP_ESTABLISHED, TCP_FIN_WAIT_1, TCP_FIN_WAIT_2, TCP_SYN_RCVD, TcpResetConnection(), and TcpSetState().

Referenced by TcpDispatcher().

VOID TcpOnAppClose (IN OUT TCP_CB * *Tcb*)

Initiate the connection close procedure, called when applications want to close the connection.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References GET_RCV_DATASIZE, TCP_CLOSE_WAIT, TCP_CLOSED, TCP_ESTABLISHED, TCP_FIN_WAIT_1, TCP_LAST_ACK, TCP_LISTEN, TCP_SYN_RCVD, TCP_SYN_SENT, TcpClose(), TcpResetConnection(), TcpSetState(), and TcpToSendData().

Referenced by TcpDispatcher().

VOID TcpOnAppConnect (IN OUT TCP_CB * *Tcb*)

Initialize an active connection.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB that wants to initiate a connection.

References TCP_SYN_SENT, TCP_TIMER_CONNECT, TcpInitTcbLocal(), TcpSetState(), TcpSetTimer(), and TcpToSendData().

Referenced by TcpDispatcher().

VOID TcpOnAppConsume (IN TCP_CB * *Tcb*)

Application has consumed some data. Check whether to send a window update ack or a delayed ack.

Parameters:

`[in]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References TCP_ESTABLISHED, TcpRcvWinNow(), TcpRcvWinOld(), and TcpSendAck().

Referenced by TcpDispatcher().

INTN TcpOnAppSend (IN OUT TCP_CB * *Tcb*)

Check whether the application's newly delivered data can be sent out.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

Return values:

0 The data has been sent out successfully.

-1 The Tcb is not in a state that data is permitted to be sent out.

References TCP_CLOSE_WAIT, TCP_CLOSED, TCP_CLOSING, TCP_ESTABLISHED, TCP_FIN_WAIT_1, TCP_FIN_WAIT_2, TCP_LAST_ACK, TCP_LISTEN, TCP_SYN_RCVD, TCP_SYN_SENT, TCP_TIME_WAIT, and TcpToSendData().

Referenced by TcpDispatcher().

VOID TcpResetConnection (IN TCP_CB * *Tcb*)

Reset the connection related with Tcb.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of the connection to be reset.

References TCP_FLG_RST, TCP_MAX_HEAD, TcpChecksum(), and TcpSendIpPacket().

Referenced by TcpConnectTimeout(), TcpOnAppAbort(), and TcpOnAppClose().

VOID TcpSetState (IN TCP_CB * *Tcb*,
IN UINT8 *State*
)

Set the Tcb's state.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.
[in] *State* The state to be set.

References mTcpStateName, SockConnClosed(), SockConnEstablished(), TCP_CLOSED, TCP_ESTABLISHED, and TcpInstallDevicePath().

Referenced by TcpClose(), TcpConfigurePcb(), TcpDeliverData(), TcpInput(), TcpOnAppAbort(), TcpOnAppClose(), and TcpOnAppConnect().

Variable Documentation

TCP_SEQNO mTcpGlobalIss = TCP_BASE_ISS

Referenced by TcpDriverEntryPoint(), TcpGetIss(), and TcpTickingDpc().

LIST_ENTRY mTcpListenQue

Initial value:

```
{
  &mTcpListenQue,
  &mTcpListenQue
}
```

Referenced by TcpFindTcbByPeer(), TcpInsertTcb(), and TcpLocateListenTcb().

LIST_ENTRY mTcpRunQue

Initial value:

```
 {
  &mTcpRunQue,
  &mTcpRunQue
}
```

Referenced by TcpFindTcbByPeer(), TcpInsertTcb(), TcpLocateTcb(), and TcpTickingDpc().

CHAR16* mTcpStateName[]

Initial value:

```
 {
  L"TCP_CLOSED",
  L"TCP_LISTEN",
  L"TCP_SYN_SENT",
  L"TCP_SYN_RCVD",
  L"TCP_ESTABLISHED",
  L"TCP_FIN_WAIT_1",
  L"TCP_FIN_WAIT_2",
  L"TCP_CLOSING",
  L"TCP_TIME_WAIT",
  L"TCP_CLOSE_WAIT",
  L"TCP_LAST_ACK"
}
```

Referenced by TcpSetState(), and TcpToSendData().

Generated on Mon Sep 28 08:49:56 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/TcpDxe/TcpOption.c File Reference

Functions

UINT16 TcpGetUint16 (IN UINT8 *Buf)
UINT32 TcpGetUint32 (IN UINT8 *Buf)
VOID TcpPutUint32 (OUT UINT8 *Buf, IN UINT32 Data)
UINT8 TcpComputeScale (IN TCP_CB *Tcb)
UINT16 TcpSynBuildOption (IN TCP_CB *Tcb, IN NET_BUF *Nbuf)
UINT16 TcpBuildOption (IN TCP_CB *Tcb, IN NET_BUF *Nbuf)
INTN TcpParseOption (IN TCP_HEAD *Tcp, IN OUT TCP_OPTION *Option)
UINT32 TcpPawsOK (IN TCP_CB *Tcb, IN UINT32 TSVal)

Detailed Description

Routines to process TCP option.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

UINT16 TcpBuildOption (IN TCP_CB * *Tcb*,
IN NET_BUF * *Nbuf*
)

Build the TCP option in synchronized states.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.
[in] *Nbuf* Pointer to the buffer to store the options.

Returns:

The total length of the TCP option field.

References mTcpTick, TCP_CTRL_SND_TS, TCP_FLG_ON, TCP_FLG_RST, TCP_OPTION_TS_ALIGNED_LEN, TCP_OPTION_TS_FAST, TcpPutUint32(), and TCPSEG_NETBUF.

Referenced by TcpTransmitSegment().

UINT8 TcpComputeScale (IN TCP_CB * *Tcb*)

Compute the window scale value according to the given buffer size.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.

Returns:

The scale value.

References GET_RCV_BUFFSIZE, TCP_OPTION_MAX_WIN, and TCP_OPTION_MAX_WS.

Referenced by TcpInitTcbPeer(), and TcpSynBuildOption().

UINT16 TcpGetUint16 (IN UINT8 * *Buf*)

Get a UINT16 value from buffer.

Parameters:

[in] *Buf* Pointer to input buffer.

Returns:

The UINT16 value obtained from the buffer.

Referenced by TcpParseOption().

UINT32 TcpGetUint32 (IN UINT8 * *Buf*)

Get a UINT32 value from buffer.

Parameters:

[in] *Buf* Pointer to input buffer.

Returns:

The UINT32 value obtained from the buffer.

Referenced by TcpParseOption().

INTN TcpParseOption (IN TCP_HEAD * *Tcp*,
IN OUT TCP_OPTION * *Option*
)

Parse the supported options.

Parameters:

[in] *Tcp* Pointer to the TCP_CB of this TCP instance.

[in,out] *Option* Pointer to the TCP_OPTION used to store the successfully pasrsed options.

Return values:

0 The options are successfully pasrsed.

-1 Ilegal option was found.

References TCP_OPTION_EOP, TCP_OPTION_MSS, TCP_OPTION_MSS_LEN, TCP_OPTION_NOP, TCP_OPTION_RCVD_MSS, TCP_OPTION_RCVD_TS, TCP_OPTION_RCVD_WS, TCP_OPTION_TS, TCP_OPTION_TS_ALIGNED_LEN, TCP_OPTION_TS_FAST, TCP_OPTION_TS_LEN, TCP_OPTION_WS, TCP_OPTION_WS_LEN, TCP_SET_FLG, TcpGetUint16(), and TcpGetUint32().

Referenced by TcpInput().

```
UINT32 TcpPawsOK ( IN TCP_CB *   Tcb,
                   IN UINT32     TSVal
                 )
```

Check the segment against PAWS.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.
[in] *TSVal* The timestamp value.

Return values:

1 The segment passed the PAWS check.
0 The segment failed to pass the PAWS check.

References mTcpTick, TCP_PAWS_24DAY, and TCP_TIME_LT.

```
VOID TcpPutUint32 ( OUT UINT8 *   Buf,
                    IN UINT32     Data
                  )
```

Put a UINT32 value in buffer.

Parameters:

[out] *Buf* Pointer to the buffer.
[in] *Data* The UINT32 Date to put in the buffer.

Referenced by TcpBuildOption(), and TcpSynBuildOption().

```
UINT16 TcpSynBuildOption ( IN TCP_CB *     Tcb,
                           IN NET_BUF *    Nbuf
                         )
```

Build the TCP option in three-way handshake.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.
[in] *Nbuf* Pointer to the buffer to store the options.

Returns:

The total length of the TCP option field.

References mTcpTick, TCP_CTRL_NO_TS, TCP_CTRL_NO_WS, TCP_CTRL_RCVD_TS, TCP_CTRL_RCVD_WS, TCP_FLG_ACK, TCP_FLG_ON, TCP_OPTION_MSS_FAST, TCP_OPTION_MSS_LEN, TCP_OPTION_TS_ALIGNED_LEN, TCP_OPTION_TS_FAST, TCP_OPTION_WS_ALIGNED_LEN, TCP_OPTION_WS_FAST, TcpComputeScale(), TcpPutUint32(), and TCPSEG_NETBUF.

Referenced by TcpTransmitSegment().

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/TcpDxe/TcpOption.h File Reference

Data Structures

struct _TCP_OPTION

Defines

"memItemRight" valign="bottom">TCP_OPTION_EOP 0
End Of oPtion.
#define TCP_OPTION_NOP 1
No-Option.
#define TCP_OPTION_MSS 2
Maximum Segment Size.
#define TCP_OPTION_WS 3
Window scale.
#define TCP_OPTION_TS 8
Timestamp.
#define TCP_OPTION_MSS_LEN 4
Length of MSS option.
#define TCP_OPTION_WS_LEN 3
Length of window scale option.
#define TCP_OPTION_TS_LEN 10
Length of timestamp option.
#define TCP_OPTION_WS_ALIGNED_LEN 4
Length of window scale option, aligned.
#define TCP_OPTION_TS_ALIGNED_LEN 12
Length of timestamp option, aligned.
#define TCP_OPTION_TS_FAST
#define TCP_OPTION_WS_FAST
#define TCP_OPTION_MSS_FAST ((TCP_OPTION_MSS << 24) | (TCP_OPTION_MSS_LEN << 16))
#define TCP_OPTION_RCVD_MSS 0x01
#define TCP_OPTION_RCVD_WS 0x02
#define TCP_OPTION_RCVD_TS 0x04
#define TCP_OPTION_MAX_WS 14
Maxium window scale value.
#define TCP_OPTION_MAX_WIN 0xffff
Max window size in TCP header.

Typedefs

typedef struct _TCP_OPTION TCP_OPTION

Functions

UINT8 TcpComputeScale (IN TCP_CB *Tcb)
UINT16 TcpSynBuildOption (IN TCP_CB *Tcb, IN NET_BUF *Nbuf)
UINT16 TcpBuildOption (IN TCP_CB *Tcb, IN

NET_BUF *Nbuf)
INTN TcpParseOption (IN TCP_HEAD *Tcp, IN OUT TCP_OPTION *Option)
UINT32 TcpPawsOK (IN TCP_CB *Tcb, IN UINT32 TSVal)

Detailed Description

Tcp option's routine header file.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

End Of oPtion.

Referenced by TcpParseOption().

#define TCP_OPTION_MAX_WIN 0xffff

Max window size in TCP header.

Referenced by TcpComputeScale().

#define TCP_OPTION_MAX_WS 14

Maxium window scale value.

Referenced by TcpComputeScale().

#define TCP_OPTION_MSS 2

Maximum Segment Size.

Referenced by TcpParseOption().

#define TCP_OPTION_MSS_FAST ((TCP_OPTION_MSS << 24) | (TCP_OPTION_MSS_LEN << 16))

Referenced by TcpSynBuildOption().

#define TCP_OPTION_MSS_LEN 4

Length of MSS option.

Referenced by TcpParseOption(), and TcpSynBuildOption().

#define TCP_OPTION_NOP 1
No-Option.

Referenced by TcpParseOption().

#define TCP_OPTION_RCVD_MSS 0x01

Referenced by TcpInitTcbPeer(), and TcpParseOption().

#define TCP_OPTION_RCVD_TS 0x04

Referenced by TcpInitTcbPeer(), TcpInput(), and TcpParseOption().

#define TCP_OPTION_RCVD_WS 0x02

Referenced by TcpInitTcbPeer(), and TcpParseOption().

#define TCP_OPTION_TS 8
Timestamp.

Referenced by TcpParseOption().

#define TCP_OPTION_TS_ALIGNED_LEN 12
Length of timestamp option, aligned.

Referenced by TcpBuildOption(), TcpInitTcbPeer(), TcpParseOption(), and TcpSynBuildOption().

#define TCP_OPTION_TS_FAST
Value:

```
((TCP_OPTION_NOP << 24) | \
                            (TCP_OPTION_NOP << 16) | \
                            (TCP_OPTION_TS << 8)   | \
                            (TCP_OPTION_TS_LEN))
```

Referenced by TcpBuildOption(), TcpParseOption(), and TcpSynBuildOption().

#define TCP_OPTION_TS_LEN 10
Length of timestamp option.

Referenced by TcpParseOption().

#define TCP_OPTION_WS 3
Window scale.

Referenced by TcpParseOption().

#define TCP_OPTION_WS_ALIGNED_LEN 4
Length of window scale option, aligned.

Referenced by TcpSynBuildOption().

#define TCP_OPTION_WS_FAST
Value:

```
((TCP_OPTION_NOP << 24) | \
                              (TCP_OPTION_WS << 16)  | \
                              (TCP_OPTION_WS_LEN << 8))
```

Referenced by TcpSynBuildOption().

#define TCP_OPTION_WS_LEN 3
Length of window scale option.

Referenced by TcpParseOption().

Typedef Documentation

typedef struct _TCP_OPTION TCP_OPTION
The structure to store the parse option value. ParseOption only parses the options, doesn't process them.

Function Documentation

UINT16 TcpBuildOption (IN TCP_CB * *Tcb*,
IN NET_BUF * *Nbuf*
)
Build the TCP option in synchronized states.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.
[in] *Nbuf* Pointer to the buffer to store the options.

Returns:

The total length of the TCP option field.

References mTcpTick, TCP_CTRL_SND_TS, TCP_FLG_ON, TCP_FLG_RST, TCP_OPTION_TS_ALIGNED_LEN, TCP_OPTION_TS_FAST, TcpPutUint32(), and TCPSEG_NETBUF.

Referenced by TcpTransmitSegment().

UINT8 TcpComputeScale (IN TCP_CB * *Tcb*)

Compute the window scale value according to the given buffer size.

Parameters:

`[in]` *Tcb* Pointer to the TCP_CB of this TCP instance.

Returns:

The scale value.

References GET_RCV_BUFFSIZE, TCP_OPTION_MAX_WIN, and TCP_OPTION_MAX_WS.

Referenced by TcpInitTcbPeer(), and TcpSynBuildOption().

INTN TcpParseOption (IN TCP_HEAD * *Tcp*,
IN OUT TCP_OPTION * *Option*
)

Parse the supported options.

Parameters:

`[in]` *Tcp* Pointer to the TCP_CB of this TCP instance.

`[in,out]` *Option* Pointer to the TCP_OPTION used to store the successfully pasrsed options.

Return values:

0 The options successfully pasrsed.

-1 Ilegal option was found.

Parse the supported options.

Parameters:

`[in]` *Tcp* Pointer to the TCP_CB of this TCP instance.

`[in,out]` *Option* Pointer to the TCP_OPTION used to store the successfully pasrsed options.

Return values:

0 The options are successfully pasrsed.

-1 Ilegal option was found.

References TCP_OPTION_EOP, TCP_OPTION_MSS, TCP_OPTION_MSS_LEN, TCP_OPTION_NOP, TCP_OPTION_RCVD_MSS, TCP_OPTION_RCVD_TS, TCP_OPTION_RCVD_WS, TCP_OPTION_TS, TCP_OPTION_TS_ALIGNED_LEN, TCP_OPTION_TS_FAST, TCP_OPTION_TS_LEN, TCP_OPTION_WS, TCP_OPTION_WS_LEN, TCP_SET_FLG, TcpGetUint16(), and TcpGetUint32().

Referenced by TcpInput().

UINT32 TcpPawsOK (IN TCP_CB * *Tcb*,
IN UINT32 *TSVal*
)

Check the segment against PAWS.

Parameters:

`[in]` *Tcb* Pointer to the TCP_CB of this TCP instance.

[in] *TSVal* The timestamp value.

Return values:

1 The segment passed the PAWS check.

0 The segment failed to pass the PAWS check.

References mTcpTick, TCP_PAWS_24DAY, and TCP_TIME_LT.

```
UINT16 TcpSynBuildOption ( IN TCP_CB *     Tcb,
                           IN NET_BUF *    Nbuf
                         )
```

Build the TCP option in three-way handshake.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.

[in] *Nbuf* Pointer to the buffer to store the options.

Returns:

The total length of the TCP option field.

References mTcpTick, TCP_CTRL_NO_TS, TCP_CTRL_NO_WS, TCP_CTRL_RCVD_TS, TCP_CTRL_RCVD_WS, TCP_FLG_ACK, TCP_FLG_ON, TCP_OPTION_MSS_FAST, TCP_OPTION_MSS_LEN, TCP_OPTION_TS_ALIGNED_LEN, TCP_OPTION_TS_FAST, TCP_OPTION_WS_ALIGNED_LEN, TCP_OPTION_WS_FAST, TcpComputeScale(), TcpPutUint32(), and TCPSEG_NETBUF.

Referenced by TcpTransmitSegment().

Generated on Mon Sep 28 08:49:56 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/TcpDxe/TcpOutput.c File Reference

Functions

UINT32 TcpRcvWinOld (IN TCP_CB *Tcb)
UINT32 TcpRcvWinNow (IN TCP_CB *Tcb)
UINT16 TcpComputeWnd (IN OUT TCP_CB *Tcb, IN BOOLEAN Syn)
TCP_SEQNO TcpGetMaxSndNxt (IN TCP_CB *Tcb)
UINT32 TcpDataToSend (IN TCP_CB *Tcb, IN INTN Force)
INTN TcpTransmitSegment (IN OUT TCP_CB *Tcb, IN NET_BUF *Nbuf)
NET_BUF * TcpGetSegmentSndQue (IN TCP_CB *Tcb, IN TCP_SEQNO Seq, IN UINT32 Len)
NET_BUF * TcpGetSegmentSock (IN TCP_CB *Tcb, IN TCP_SEQNO Seq, IN UINT32 Len)
NET_BUF * TcpGetSegment (IN TCP_CB *Tcb, IN TCP_SEQNO Seq, IN UINT32 Len)
INTN TcpRetransmit (IN TCP_CB *Tcb, IN TCP_SEQNO Seq)
INTN TcpCheckSndQue (IN LIST_ENTRY *Head)
INTN TcpToSendData (IN OUT TCP_CB *Tcb, IN INTN Force)
VOID TcpSendAck (IN OUT TCP_CB *Tcb)
INTN TcpSendZeroProbe (IN OUT TCP_CB *Tcb)
VOID TcpToSendAck (IN OUT TCP_CB *Tcb)
INTN TcpSendReset (IN TCP_CB *Tcb, IN TCP_HEAD *Head, IN INT32 Len, IN EFI_IP_ADDRESS *Local, IN EFI_IP_ADDRESS *Remote, IN UINT8 Version)
INTN TcpVerifySegment (IN NET_BUF *Nbuf)

Variables

UINT8 mTcpOutFlag []

Detailed Description

TCP output process routines.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

INTN TcpCheckSndQue (IN LIST_ENTRY * *Head*)

Verify that all the segments in SndQue are in good shape.

Parameters:

`[in]` *Head* Pointer to the head node of the SndQue.

Return values:

0 At least one segment is broken.
1 All segments in the specific queue are in good shape.

References TCPSEG_NETBUF, and TcpVerifySegment().

Referenced by TcpToSendData().

```
UINT16 TcpComputeWnd ( IN OUT TCP_CB *   Tcb,
                       IN BOOLEAN        Syn
                     )
```

Compute the value to fill in the window size field of the outgoing segment.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.
`[in]` *Syn* The flag to indicate whether the outgoing segment is a SYN segment.

Returns:

The value of the local receive window size used to fill the outgoing segment.

References GET_RCV_BUFFSIZE, and TcpRcvWinNow().

Referenced by TcpTransmitSegment().

```
UINT32 TcpDataToSend ( IN TCP_CB *   Tcb,
                       IN INTN       Force
                     )
```

Compute how much data to send.

Parameters:

`[in]` *Tcb* Pointer to the TCP_CB of this TCP instance.
`[in]` *Force* If TRUE, to ignore the sender's SWS avoidance algorithm and send out data by force.

Returns:

The length of the data can be sent. If 0, no data can be sent.

References GET_SND_DATASIZE, TCP_CTRL_NO_NAGLE, TCP_FLG_ON, TCP_SEQ_GT, TCP_SUB_SEQ, TCP_TIMER_ON, TCP_TIMER_REXMIT, TcpGetMaxSndNxt(), and TcpSetProbeTimer().

Referenced by TcpProbeTimeout(), and TcpToSendData().

```
TCP_SEQNO TcpGetMaxSndNxt ( IN TCP_CB *   Tcb   )
```

Get the maximum SndNxt.

Parameters:

`[in]` *Tcb* Pointer to the TCP_CB of this TCP instance.

Returns:

The sequence number of the maximum SndNxt.

References TCP_SEQ_GEQ, and TCPSEG_NETBUF.

Referenced by TcpDataToSend(), TcpDispatcher(), TcpGetSegment(), and TcpToSendData().

```
NET_BUF* TcpGetSegment ( IN TCP_CB *      Tcb,
                         IN TCP_SEQNO     Seq,
                         IN UINT32        Len
                       )
```

Get a segment starting from sequence Seq of a maximum length of Len.

Parameters:

- [in] *Tcb* Pointer to the TCP_CB of this TCP instance.
- [in] *Seq* The sequence number of the segment.
- [in] *Len* The maximum length of the segment.

Returns:

Pointer to the segment. If NULL, some error occurred.

References TCP_SEQ_LT, TcpGetMaxSndNxt(), TcpGetSegmentSndQue(), TcpGetSegmentSock(), and TcpVerifySegment().

Referenced by TcpToSendData().

```
NET_BUF* TcpGetSegmentSndQue ( IN TCP_CB *      Tcb,
                               IN TCP_SEQNO     Seq,
                               IN UINT32        Len
                             )
```

Get a segment from the Tcb's SndQue.

Parameters:

- [in] *Tcb* Pointer to the TCP_CB of this TCP instance.
- [in] *Seq* The sequence number of the segment.
- [in] *Len* The maximum length of the segment.

Returns:

Pointer to the segment. If NULL, some error occurred.

References _TCP_SEG::End, _TCP_SEG::Flag, _TCP_SEG::Seq, TCP_CLEAR_FLG, TCP_FLG_FIN, TCP_FLG_ON, TCP_FLG_SYN, TCP_MAX_HEAD, TCP_SEQ_LEQ, TCP_SEQ_LT, TCP_SUB_SEQ, and TCPSEG_NETBUF.

Referenced by TcpGetSegment(), and TcpRetransmit().

```
NET_BUF* TcpGetSegmentSock ( IN TCP_CB *      Tcb,
                             IN TCP_SEQNO     Seq,
                             IN UINT32        Len
                           )
```

Get a segment from the Tcb's socket buffer.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.
[in] *Seq* The sequence number of the segment.
[in] *Len* The maximum length of the segment.

Returns:

Pointer to the segment. If NULL, some error occurred.

References SockDataSent(), SockGetDataToSend(), TCP_MAX_HEAD, and TCPSEG_NETBUF.

Referenced by TcpGetSegment().

UINT32 TcpRcvWinNow (IN TCP_CB * *Tcb*)

Compute the current receive window.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.

Returns:

The size of the current receive window, in bytes.

References GET_RCV_BUFFSIZE, SOCK_RCV_BUF, SockGetFreeSpace(), and TcpRcvWinOld().

Referenced by TcpComputeWnd(), TcpOnAppConsume(), and TcpToSendAck().

UINT32 TcpRcvWinOld (IN TCP_CB * *Tcb*)

Compute the sequence space left in the old receive window.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.

Returns:

The sequence space left in the old receive window.

References TCP_SEQ_GT, and TCP_SUB_SEQ.

Referenced by TcpOnAppConsume(), TcpRcvWinNow(), and TcpToSendAck().

INTN TcpRetransmit (IN TCP_CB * *Tcb*,
IN TCP_SEQNO *Seq*
)

Retransmit the segment from sequence Seq.

Parameters:

[in] *Tcb* Pointer to the TCP_CB of this TCP instance.
[in] *Seq* The sequence number of the segment to be retransmitted.

Return values:

0 Retransmission succeeded.
-1 Error condition occurred.

References TCP_SEQ_LT, TCP_SUB_SEQ, TcpGetSegmentSndQue(), TcpTransmitSegment(), and TcpVerifySegment().

Referenced by TcpFastLossRecover(), TcpFastRecover(), and TcpRexmitTimeout().

VOID TcpSendAck (IN OUT TCP_CB * *Tcb*)

Send an ACK immediately.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References _TCP_SEG::End, _TCP_SEG::Flag, _TCP_SEG::Seq, TCP_CLEAR_FLG, TCP_CTRL_ACK_NOW, TCP_FLG_ACK, TCP_MAX_HEAD, TCPSEG_NETBUF, and TcpTransmitSegment().

Referenced by TcpDeliverData(), TcpInput(), TcpOnAppConsume(), TcpTickingDpc(), and TcpToSendAck().

```
INTN TcpSendReset ( IN TCP_CB *              Tcb,
                    IN TCP_HEAD *            Head,
                    IN INT32                 Len,
                    IN EFI_IP_ADDRESS *      Local,
                    IN EFI_IP_ADDRESS *      Remote,
                    IN UINT8                 Version
                  )
```

Send a RESET segment in response to the segment received.

Parameters:

`[in]`	*Tcb*	Pointer to the TCP_CB of this TCP instance. May be NULL.
`[in]`	*Head*	TCP header of the segment that triggers the reset.
`[in]`	*Len*	Length of the segment that triggers the reset.
`[in]`	*Local*	Local IP address.
`[in]`	*Remote*	Remote peer's IP address.
`[in]`	*Version*	IP_VERSION_4 indicates TCP is running on IP4 stack, IP_VERSION_6 indicates TCP is running on IP6 stack.

Return values:

0 A reset was sent or there is no need to send it.

-1 No reset is sent.

References TCP_FLG_ACK, TCP_FLG_ON, TCP_FLG_RST, TCP_MAX_HEAD, TCP_SET_FLG, TcpChecksum(), and TcpSendIpPacket().

Referenced by TcpInput().

INTN TcpSendZeroProbe (IN OUT TCP_CB * *Tcb*)

Send a zero probe segment. It can be used by keepalive and zero window probe.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

Return values:

0 The zero probe segment was sent out successfully.

other An error condition occurred.

References _TCP_SEG::End, _TCP_SEG::Flag, _TCP_SEG::Seq, TCP_FLG_ACK, TCP_MAX_HEAD, TCPSEG_NETBUF, and TcpTransmitSegment().

Referenced by TcpKeepaliveTimeout(), and TcpProbeTimeout().

VOID TcpToSendAck (IN OUT TCP_CB * *Tcb*)

Check whether to send an ACK or delayed ACK.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References TCP_CTRL_ACK_NOW, TCP_FLG_ON, TcpRcvWinNow(), TcpRcvWinOld(), and TcpSendAck().

Referenced by TcpInput().

INTN TcpToSendData (IN OUT TCP_CB * *Tcb*,
IN INTN *Force*
)

Check whether to send data/SYN/FIN and piggyback an ACK.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

`[in]` *Force* If TRUE, ignore the sender's SWS avoidance algorithm and send out data by force.

Returns:

The number of bytes sent.

References _TCP_SEG::End, _TCP_SEG::Flag, GET_SND_DATASIZE, mTcpOutFlag, mTcpStateName, _TCP_SEG::Seq, TCP_CLEAR_FLG, TCP_CLOSED, TCP_CONGEST_OPEN, TCP_CTRL_FIN_SENT, TCP_CTRL_RTT_ON, TCP_FLG_FIN, TCP_FLG_ON, TCP_FLG_SYN, TCP_LISTEN, TCP_SEQ_GT, TCP_SEQ_LT, TCP_SET_FLG, TCP_SUB_SEQ, TCP_TIMER_ON, TCP_TIMER_REXMIT, TcpCheckSndQue(), TcpDataToSend(), TcpGetMaxSndNxt(), TcpGetSegment(), TCPSEG_NETBUF, TcpSetTimer(), TcpTransmitSegment(), and TcpVerifySegment().

Referenced by TcpInput(), TcpOnAppClose(), TcpOnAppConnect(), TcpOnAppSend(), and TcpProbeTimeout().

INTN TcpTransmitSegment (IN OUT TCP_CB * *Tcb*,
IN NET_BUF * *Nbuf*
)

Build the TCP header of the TCP segment and transmit the segment by IP.

Parameters:

[in,out] *Tcb* Pointer to the TCP_CB of this TCP instance.

[in] *Nbuf* Pointer to the buffer containing the segment to be sent out.

Return values:

0 The segment was sent out successfully.

-1 An error condition occurred.

References _TCP_SEG::End, _TCP_SEG::Flag, GET_SND_DATASIZE, _TCP_SEG::Seq, TCP_CLEAR_FLG, TCP_CTRL_SND_PSH, TCP_CTRL_SND_URG, TCP_FLG_ON, TCP_FLG_PSH, TCP_FLG_SYN, TCP_FLG_URG, TCP_SEQ_BETWEEN, TCP_SEQ_LEQ, TCP_SEQ_LT, TCP_SET_FLG, TCP_SUB_SEQ, TcpBuildOption(), TcpChecksum(), TcpComputeWnd(), TCPSEG_NETBUF, TcpSendIpPacket(), TcpSynBuildOption(), TcpVerifySegment(), and _TCP_SEG::Urg.

Referenced by TcpRetransmit(), TcpSendAck(), TcpSendZeroProbe(), and TcpToSendData().

INTN TcpVerifySegment (IN NET_BUF * *Nbuf*)

Verify that the segment is in good shape.

Parameters:

[in] *Nbuf* The buffer that contains the segment to be checked.

Return values:

0 The segment is broken.

1 The segment is in good shape.

References _TCP_SEG::End, _TCP_SEG::Flag, _TCP_SEG::Seq, TCP_FLG_FIN, TCP_FLG_ON, TCP_FLG_SYN, and TCPSEG_NETBUF.

Referenced by TcpCheckSndQue(), TcpDeliverData(), TcpGetSegment(), TcpRetransmit(), TcpToSendData(), TcpTransmitSegment(), and TcpTrimSegment().

Variable Documentation

UINT8 mTcpOutFlag[]

Initial value:

```
{
 0,
 0,
 TCP_FLG_SYN,
 TCP_FLG_SYN | TCP_FLG_ACK,
 TCP_FLG_ACK,
 TCP_FLG_FIN | TCP_FLG_ACK,
 TCP_FLG_ACK,
 TCP_FLG_ACK | TCP_FLG_FIN,
 TCP_FLG_ACK,
 TCP_FLG_ACK,
 TCP_FLG_FIN | TCP_FLG_ACK
}
```

Referenced by TcpToSendData().

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/TcpDxe/TcpProto.h File Reference

Data Structures

struct _TCP_SEG
struct _TCP_PEER
struct _TCP_CONTROL_BLOCK

Defines

"memItemRight" valign="bottom">TCP_CLOSED 0
#define TCP_LISTEN 1
#define TCP_SYN_SENT 2
#define TCP_SYN_RCVD 3
#define TCP_ESTABLISHED 4
#define TCP_FIN_WAIT_1 5
#define TCP_FIN_WAIT_2 6
#define TCP_CLOSING 7
#define TCP_TIME_WAIT 8
#define TCP_CLOSE_WAIT 9
#define TCP_LAST_ACK 10
#define TCP_FLG_FIN 0x01
#define TCP_FLG_SYN 0x02
#define TCP_FLG_RST 0x04
#define TCP_FLG_PSH 0x08
#define TCP_FLG_ACK 0x10
#define TCP_FLG_URG 0x20
#define TCP_FLG_FLAG 0x3F
#define TCP_CONNECT_REFUSED (-1)
TCP error status.
#define TCP_CONNECT_RESET (-2)
TCP error status.
#define TCP_CONNECT_CLOSED (-3)
TCP error status.
#define TCP_CONGEST_RECOVER 1
During the NewReno fast recovery.
#define TCP_CONGEST_LOSS 2
Retxmit because of retxmit time out.
#define TCP_CONGEST_OPEN 3
TCP is opening its congestion window.
#define TCP_CTRL_NO_NAGLE 0x0001
Disable Nagle algorithm.
#define TCP_CTRL_NO_KEEPALIVE 0x0002
Disable keepalive timer.
#define TCP_CTRL_NO_WS 0x0004
Disable window scale option.
#define TCP_CTRL_RCVD_WS 0x0008
Received a wnd scale option in syn.
#define TCP_CTRL_NO_TS 0x0010
Disable Timestamp option.
#define TCP_CTRL_RCVD_TS 0x0020
Received a Timestamp option in syn.

#define TCP_CTRL_SND_TS 0x0040
Send Timestamp option to remote.
#define TCP_CTRL_SND_URG 0x0080
In urgent send mode.
#define TCP_CTRL_RCVD_URG 0x0100
In urgent receive mode.
#define TCP_CTRL_SND_PSH 0x0200
In PUSH send mode.
#define TCP_CTRL_FIN_SENT 0x0400
FIN is sent.
#define TCP_CTRL_FIN_ACKED 0x0800
FIN is ACKed.
#define TCP_CTRL_TIMER_ON 0x1000
At least one of the timer is on.
#define TCP_CTRL_RTT_ON 0x2000
The RTT measurement is on.
#define TCP_CTRL_ACK_NOW 0x4000
Send the ACK now, don't delay.
#define TCP_TIMER_CONNECT 0
Connection establishment timer.
#define TCP_TIMER_REXMIT 1
Retransmit timer.
#define TCP_TIMER_PROBE 2
Window probe timer.
#define TCP_TIMER_KEEPALIVE 3
Keepalive timer.
#define TCP_TIMER_FINWAIT2 4
FIN_WAIT_2 timer.
#define TCP_TIMER_2MSL 5
TIME_WAIT timer.
#define TCP_TIMER_NUMBER 6
The total number of the TCP timer.
#define TCP_TICK 200
Every TCP tick is 200ms.
#define TCP_TICK_HZ 5
The frequence of TCP tick.
#define TCP_RTT_SHIFT 3
SRTT & RTTVAR scaled by 8.
#define TCP_RTO_MIN TCP_TICK_HZ
The minium value of RTO.
#define TCP_RTO_MAX (TCP_TICK_HZ * 60)
The maxium value of RTO.
#define TCP_FOLD_RTT 4
Timeout threshod to fold RTT.
#define TCP_MAX_LOSS 12
Default max times to retxmit.
#define TCP_KEEPALIVE_IDLE_MIN (TCP_TICK_HZ * 60 * 60 * 2)
First keepalive.
#define TCP_KEEPALIVE_PERIOD (TCP_TICK_HZ * 60)
#define TCP_MAX_KEEPALIVE 8
#define TCP_FIN_WAIT2_TIME (2 * TCP_TICK_HZ)
#define TCP_TIME_WAIT_TIME (2 * TCP_TICK_HZ)
#define TCP_PAWS_24DAY (24 * 24 * 60 * 60 *

```
TCP_TICK_HZ)
#define TCP_CONNECT_TIME (75 * TCP_TICK_HZ)
#define TCP_MAX_HEAD 192
#define TCP_RCV_BUF_SIZE (2 * 1024 * 1024)
#define TCP_RCV_BUF_SIZE_MIN (8 * 1024)
#define TCP_SND_BUF_SIZE (2 * 1024 * 1024)
#define TCP_SND_BUF_SIZE_MIN (8 * 1024)
#define TCP_BACKLOG 10
#define TCP_BACKLOG_MIN 5
#define TCP_MAX_LOSS_MIN 6
#define TCP_CONNECT_TIME_MIN (60 *
        TCP_TICK_HZ)
#define TCP_MAX_KEEPALIVE_MIN 4
#define TCP_KEEPALIVE_IDLE_MAX (TCP_TICK_HZ *
        60 * 60 * 4)
#define TCP_KEEPALIVE_PERIOD_MIN (TCP_TICK_HZ
        * 30)
#define TCP_FIN_WAIT2_TIME_MAX (4 *
        TCP_TICK_HZ)
#define TCP_TIME_WAIT_TIME_MAX (60 *
        TCP_TICK_HZ)
#define TCP_CONNECTED(state) ((state) >
        TCP_SYN_RCVD)
#define TCP_FIN_RCVD(State)
#define TCP_LOCAL_CLOSED(State)
#define TCPSEG_NETBUF(NBuf) ((TCP_SEG *)
        ((NBuf)->ProtoData))
#define TCP_SEQ_LT(SeqA, SeqB) ((INT32) ((SeqA) -
        (SeqB)) < 0)
#define TCP_SEQ_LEQ(SeqA, SeqB) ((INT32) ((SeqA) -
        (SeqB)) <= 0)
#define TCP_SEQ_GT(SeqA, SeqB) ((INT32) ((SeqB) -
        (SeqA)) < 0)
#define TCP_SEQ_GEQ(SeqA, SeqB) ((INT32) ((SeqB) -
        (SeqA)) <= 0)
#define TCP_SEQ_BETWEEN(b, m, e) ((e) - (b) >= (m) -
        (b))
#define TCP_SUB_SEQ(Seq1, Seq2) ((UINT32) ((Seq1) -
        (Seq2)))
#define TCP_FLG_ON(Value, Flag) ((BOOLEAN) (((Value)
        & (Flag)) != 0))
#define TCP_SET_FLG(Value, Flag) ((Value) |= (Flag))
#define TCP_CLEAR_FLG(Value, Flag) ((Value) &=
        ~(Flag))
#define TCP_PEER_EQUAL(Pa, Pb, Ver) (((Pa)->Port ==
        (Pb)->Port) && TcpIsIpEqual(&((Pa)->Ip),
        &((Pb)->Ip), Ver))
#define TCP_PEER_MATCH(Pa, Pb, Ver)
#define TCP_TIMER_ON(Flag, Timer) ((Flag) & (1 <<
        (Timer)))
#define TCP_SET_TIMER(Flag, Timer) ((Flag) = (UINT16)
        ((Flag) | (1 << (Timer))))
#define TCP_CLEAR_TIMER(Flag, Timer) ((Flag) =
        (UINT16) ((Flag) & (~(1 << (Timer)))))
```

```
#define TCP_TIME_LT(Ta, Tb)   ((INT32) ((Ta) - (Tb)) < 0)
#define TCP_TIME_LEQ(Ta, Tb)   ((INT32) ((Ta) - (Tb)) <=
        0)
#define TCP_SUB_TIME(Ta, Tb)   ((UINT32) ((Ta) - (Tb)))
#define TCP_MAX_WIN   0xFFFFU
```

Typedefs

```
typedef struct _TCP_SEG TCP_SEG
typedef struct _TCP_PEER TCP_PEER
typedef struct _TCP_CONTROL_BLOCK TCP_CB
```

Detailed Description

TCP protocol header file.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by TcpConfigurePcb().

```
#define TCP_BACKLOG_MIN   5
```

Referenced by TcpConfigurePcb().

```
#define TCP_CLEAR_FLG ( Value,
                        Flag     )    ((Value) &= ~(Flag))
```

Referenced by TcpConfigurePcb(), TcpFastRecover(), TcpGetSegmentSndQue(), TcpInput(), TcpRexmitTimeout(), TcpSendAck(), TcpToSendData(), TcpTransmitSegment(), TcpTrimSegment(), and TcpUpdateTimer().

```
#define TCP_CLEAR_TIMER ( Flag,
                          Timer    )    ((Flag) = (UINT16) ((Flag) & (~(1 << (Timer)))))
```

Referenced by TcpClearTimer(), and TcpTickingDpc().

#define TCP_CLOSE_WAIT 9

Referenced by TcpDeliverData(), TcpInput(), TcpOnAppAbort(), TcpOnAppClose(), and TcpOnAppSend().

#define TCP_CLOSED 0

Tcp states don't change their order. It is used as an index to mTcpOutFlag and other macros.

Referenced by TcpAttachPcb(), TcpClose(), TcpConfigurePcb(), TcpFlushPcb(), TcpIcmpInput(), TcpInput(), TcpInsertTcb(), TcpOnAppAbort(), TcpOnAppClose(), TcpOnAppSend(), TcpSetState(), TcpTickingDpc(), and TcpToSendData().

#define TCP_CLOSING 7

Referenced by TcpDeliverData(), TcpInput(), and TcpOnAppSend().

#define TCP_CONGEST_LOSS 2

Retxmit because of retxmit time out.

Referenced by TcpInput(), and TcpRexmitTimeout().

#define TCP_CONGEST_OPEN 3

TCP is opening its congestion window.

Referenced by TcpConfigurePcb(), TcpFastLossRecover(), TcpFastRecover(), TcpInput(), and TcpToSendData().

#define TCP_CONGEST_RECOVER 1

During the NewReno fast recovery.

Referenced by TcpFastRecover(), and TcpInput().

#define TCP_CONNECT_CLOSED (-3)

TCP error status.

#define TCP_CONNECT_REFUSED (-1)

TCP error status.

#define TCP_CONNECT_RESET (-2)

TCP error status.

#define TCP_CONNECT_TIME (75 * TCP_TICK_HZ)

Referenced by TcpConfigurePcb().

#define TCP_CONNECT_TIME_MIN (60 * TCP_TICK_HZ)

Referenced by TcpConfigurePcb().

#define TCP_CONNECTED (state) ((state) > TCP_SYN_RCVD)
TCP_CONNECTED: both ends have synchronized their ISN.

Referenced by TcpConnectTimeout(), and TcpDeliverData().

#define TCP_CTRL_ACK_NOW 0x4000
Send the ACK now, don't delay.

Referenced by TcpDeliverData(), TcpInput(), TcpQueueData(), TcpSendAck(), and TcpToSendAck().

#define TCP_CTRL_FIN_ACKED 0x0800
FIN is ACKed.

Referenced by TcpDeliverData(), and TcpInput().

#define TCP_CTRL_FIN_SENT 0x0400
FIN is sent.

Referenced by TcpInput(), and TcpToSendData().

#define TCP_CTRL_NO_KEEPALIVE 0x0002
Disable keepalive timer.

Referenced by TcpConfigurePcb(), and TcpSetKeepaliveTimer().

#define TCP_CTRL_NO_NAGLE 0x0001
Disable Nagle algorithm.

Referenced by Tcp4GetMode(), Tcp6GetMode(), TcpConfigurePcb(), and TcpDataToSend().

#define TCP_CTRL_NO_TS 0x0010
Disable Timestamp option.

Referenced by Tcp4GetMode(), Tcp6GetMode(), TcpConfigurePcb(), TcpInitTcbPeer(), and TcpSynBuildOption().

#define TCP_CTRL_NO_WS 0x0004
Disable window scale option.

Referenced by Tcp4GetMode(), Tcp6GetMode(), TcpConfigurePcb(), TcpInitTcbPeer(), and TcpSynBuildOption().

#define TCP_CTRL_RCVD_TS 0x0020
Received a Timestamp option in syn.

Referenced by TcpInitTcbPeer(), and TcpSynBuildOption().

#define TCP_CTRL_RCVD_URG 0x0100
In urgent receive mode.

Referenced by TcpDeliverData(), and TcpInput().

#define TCP_CTRL_RCVD_WS 0x0008
Received a wnd scale option in syn.

Referenced by TcpInitTcbPeer(), and TcpSynBuildOption().

#define TCP_CTRL_RTT_ON 0x2000
The RTT measurement is on.

Referenced by TcpFastRecover(), TcpInput(), TcpRexmitTimeout(), TcpTickingDpc(), and TcpToSendData().

#define TCP_CTRL_SND_PSH 0x0200
In PUSH send mode.

Referenced by TcpDispatcher(), and TcpTransmitSegment().

#define TCP_CTRL_SND_TS 0x0040
Send Timestamp option to remote.

Referenced by TcpBuildOption(), and TcpInitTcbPeer().

#define TCP_CTRL_SND_URG 0x0080
In urgent send mode.

Referenced by TcpDispatcher(), TcpInput(), and TcpTransmitSegment().

#define TCP_CTRL_TIMER_ON 0x1000
At least one of the timer is on.

Referenced by TcpTickingDpc(), and TcpUpdateTimer().

#define TCP_ESTABLISHED 4

Referenced by TcpDeliverData(), TcpInput(), TcpOnAppAbort(), TcpOnAppClose(), TcpOnAppConsume(), TcpOnAppSend(), and TcpSetState().

#define TCP_FIN_RCVD (State)

Value:

```
( \
    ((State) == TCP_CLOSE_WAIT) || \
    ((State) == TCP_LAST_ACK) || \
    ((State) == TCP_CLOSING) || \
    ((State) == TCP_TIME_WAIT) \
  )
```

Referenced by TcpDeliverData(), and TcpInput().

#define TCP_FIN_WAIT2_TIME (2 * TCP_TICK_HZ)

Referenced by TcpConfigurePcb().

#define TCP_FIN_WAIT2_TIME_MAX (4 * TCP_TICK_HZ)

Referenced by TcpConfigurePcb().

#define TCP_FIN_WAIT_1 5

Referenced by TcpDeliverData(), TcpInput(), TcpOnAppAbort(), TcpOnAppClose(), and TcpOnAppSend().

#define TCP_FIN_WAIT_2 6

Referenced by TcpDeliverData(), TcpInput(), TcpOnAppAbort(), and TcpOnAppSend().

#define TCP_FLG_ACK 0x10

Referenced by TcpInitTcbPeer(), TcpInput(), TcpSendAck(), TcpSendReset(), TcpSendZeroProbe(), and TcpSynBuildOption().

#define TCP_FLG_FIN 0x01

Flags in the TCP header

Referenced by TcpDeliverData(), TcpFormatNetbuf(), TcpGetSegmentSndQue(), TcpInput(), TcpToSendData(), TcpTrimSegment(), and TcpVerifySegment().

#define TCP_FLG_FLAG 0x3F

#define TCP_FLG_ON (Value,
Flag) ((BOOLEAN) (((Value) & (Flag)) != 0))

Referenced by Tcp4GetMode(), Tcp6GetMode(), TcpBuildOption(), TcpDataToSend(), TcpDeliverData(), TcpFormatNetbuf(), TcpGetSegmentSndQue(), TcpInitTcbPeer(), TcpInput(), TcpSendReset(),

TcpSetKeepaliveTimer(), TcpSynBuildOption(), TcpTickingDpc(), TcpToSendAck(), TcpToSendData(), TcpTransmitSegment(), TcpTrimSegment(), and TcpVerifySegment().

#define TCP_FLG_PSH 0x08

Referenced by TcpDeliverData(), and TcpTransmitSegment().

#define TCP_FLG_RST 0x04

Referenced by TcpBuildOption(), TcpInput(), TcpResetConnection(), and TcpSendReset().

#define TCP_FLG_SYN 0x02

Referenced by TcpFormatNetbuf(), TcpGetSegmentSndQue(), TcpInitTcbPeer(), TcpInput(), TcpToSendData(), TcpTransmitSegment(), TcpTrimSegment(), and TcpVerifySegment().

#define TCP_FLG_URG 0x20

Referenced by TcpInput(), TcpTransmitSegment(), and TcpTrimSegment().

#define TCP_FOLD_RTT 4
Timeout threshod to fold RTT.

Referenced by TcpBackoffRto().

#define TCP_KEEPALIVE_IDLE_MAX (TCP_TICK_HZ * 60 * 60 * 4)

Referenced by TcpConfigurePcb().

#define TCP_KEEPALIVE_IDLE_MIN (TCP_TICK_HZ * 60 * 60 * 2)
First keepalive.

Referenced by TcpConfigurePcb().

#define TCP_KEEPALIVE_PERIOD (TCP_TICK_HZ * 60)

Referenced by TcpConfigurePcb().

#define TCP_KEEPALIVE_PERIOD_MIN (TCP_TICK_HZ * 30)

Referenced by TcpConfigurePcb().

#define TCP_LAST_ACK 10

Referenced by TcpDeliverData(), TcpInput(), TcpOnAppClose(), and TcpOnAppSend().

#define TCP_LISTEN 1

Referenced by Tcp4GetMode(), Tcp6GetMode(), TcpConfigurePcb(), TcpInput(), TcpInsertTcb(), TcpOnAppClose(), TcpOnAppSend(), and TcpToSendData().

#define TCP_LOCAL_CLOSED (State)

Value:

```
( \
    ((State) == TCP_FIN_WAIT_1) || \
    ((State) == TCP_FIN_WAIT_2) || \
    ((State) == TCP_CLOSING) || \
    ((State) == TCP_TIME_WAIT) || \
    ((State) == TCP_LAST_ACK) \
  )
```

Referenced by TcpInput().

#define TCP_MAX_HEAD 192

Referenced by TcpGetSegmentSndQue(), TcpGetSegmentSock(), TcpResetConnection(), TcpSendAck(), TcpSendReset(), and TcpSendZeroProbe().

#define TCP_MAX_KEEPALIVE 8

Referenced by TcpConfigurePcb().

#define TCP_MAX_KEEPALIVE_MIN 4

Referenced by TcpConfigurePcb().

#define TCP_MAX_LOSS 12

Default max times to retxmit.

Referenced by TcpConfigurePcb().

#define TCP_MAX_LOSS_MIN 6

Referenced by TcpConfigurePcb().

#define TCP_MAX_WIN 0xFFFFU

Referenced by TcpInput().

#define TCP_PAWS_24DAY (24 * 24 * 60 * 60 * TCP_TICK_HZ)

Referenced by TcpPawsOK().

#define TCP_PEER_EQUAL (Pa, Pb, Ver) (((Pa)->Port == (Pb)->Port) && TcpIsIpEqual(&((Pa)->Ip), &((Pb)->Ip), Ver))

Referenced by TcpInsertTcb(), and TcpLocateTcb().

#define TCP_PEER_MATCH (Pa, Pb, Ver)

Value:

```
( \
    (((Pb)->Port == 0) || ((Pb)->Port == (Pa)->Port)) && \
    (TcpIsIpZero (&((Pb)->Ip), Ver) || TcpIsIpEqual (&((Pb)->Ip), &((Pa)->Ip), Ver)) \
  )
```

Referenced by TcpLocateListenTcb().

#define TCP_RCV_BUF_SIZE (2 * 1024 * 1024)

Referenced by TcpConfigurePcb().

#define TCP_RCV_BUF_SIZE_MIN (8 * 1024)

Referenced by TcpConfigurePcb().

#define TCP_RTO_MAX (TCP_TICK_HZ * 60)

The maxium value of RTO.

Referenced by TcpBackoffRto(), TcpComputeRtt(), and TcpSetProbeTimer().

#define TCP_RTO_MIN TCP_TICK_HZ

The minium value of RTO.

Referenced by TcpBackoffRto(), TcpComputeRtt(), and TcpSetProbeTimer().

#define TCP_RTT_SHIFT 3

SRTT & RTTVAR scaled by 8.

Referenced by TcpComputeRtt().

```
#define TCP_SEQ_BETWEEN ( b,
                          m,
                          e    )    ((e) - (b) >= (m) - (b))
```

Referenced by TcpTransmitSegment().

```
#define TCP_SEQ_GEQ ( SeqA,
                      SeqB    )    ((INT32) ((SeqB) - (SeqA)) <= 0)
```

Referenced by TcpAdjustSndQue(), TcpFastLossRecover(), TcpFastRecover(), and TcpGetMaxSndNxt().

```
#define TCP_SEQ_GT ( SeqA,
                     SeqB    )    ((INT32) ((SeqB) - (SeqA)) < 0)
```

Referenced by TcpDataToSend(), TcpDeliverData(), TcpFastLossRecover(), TcpInput(), TcpRcvWinOld(), TcpToSendData(), and TcpTrimSegment().

```
#define TCP_SEQ_LEQ ( SeqA,
                      SeqB    )    ((INT32) ((SeqA) - (SeqB)) <= 0)
```

Referenced by TcpAdjustSndQue(), TcpDeliverData(), TcpGetSegmentSndQue(), TcpIcmpInput(), TcpInput(), TcpQueueData(), TcpSeqAcceptable(), TcpTransmitSegment(), and TcpTrimSegment().

```
#define TCP_SEQ_LT ( SeqA,
                     SeqB    )    ((INT32) ((SeqA) - (SeqB)) < 0)
```

Referenced by TcpGetSegment(), TcpGetSegmentSndQue(), TcpIcmpInput(), TcpInput(), TcpQueueData(), TcpRetransmit(), TcpSeqAcceptable(), TcpToSendData(), TcpTransmitSegment(), and TcpTrimSegment().

```
#define TCP_SET_FLG ( Value,
                      Flag    )    ((Value) |= (Flag))
```

Referenced by TcpConfigurePcb(), TcpDeliverData(), TcpDispatcher(), TcpInitTcbPeer(), TcpInput(), TcpParseOption(), TcpQueueData(), TcpSendReset(), TcpToSendData(), TcpTransmitSegment(), and TcpUpdateTimer().

```
#define TCP_SET_TIMER ( Flag,
                        Timer    )    ((Flag) = (UINT16) ((Flag) | (1 << (Timer))))
```

Referenced by TcpSetTimer().

```
#define TCP_SND_BUF_SIZE   (2 * 1024 * 1024)
```

Referenced by TcpConfigurePcb().

```
#define TCP_SND_BUF_SIZE_MIN   (8 * 1024)
```

Referenced by TcpConfigurePcb().

```
#define TCP_SUB_SEQ  ( Seq1,
                       Seq2    )     ((UINT32) ((Seq1) - (Seq2)))
```

Referenced by TcpDataToSend(), TcpDeliverData(), TcpFastRecover(), TcpGetSegmentSndQue(), TcpRcvWinOld(), TcpRetransmit(), TcpRexmitTimeout(), TcpToSendData(), TcpTransmitSegment(), and TcpTrimSegment().

```
#define TCP_SUB_TIME  ( Ta,
                        Tb    )     ((UINT32) ((Ta) - (Tb)))
```

Referenced by TcpInput(), and TcpUpdateTimer().

```
#define TCP_SYN_RCVD   3
```

Referenced by TcpConnectTimeout(), TcpDeliverData(), TcpInput(), TcpInsertTcb(), TcpOnAppAbort(), TcpOnAppClose(), and TcpOnAppSend().

```
#define TCP_SYN_SENT   2
```

Referenced by TcpInput(), TcpInsertTcb(), TcpOnAppClose(), TcpOnAppConnect(), and TcpOnAppSend().

```
#define TCP_TICK   200
```

Every TCP tick is 200ms.

```
#define TCP_TICK_HZ   5
```

The frequence of TCP tick.

Referenced by Tcp4GetMode(), Tcp6GetMode(), TcpConfigurePcb(), and TcpCreateTimer().

```
#define TCP_TIME_LEQ  ( Ta,
                        Tb    )     ((INT32) ((Ta) - (Tb)) <= 0)
```

Referenced by TcpTickingDpc().

```
#define TCP_TIME_LT  ( Ta,
                       Tb    )     ((INT32) ((Ta) - (Tb)) < 0)
```

Referenced by TcpPawsOK(), and TcpUpdateTimer().

#define TCP_TIME_WAIT 8

Referenced by TcpDeliverData(), TcpInput(), and TcpOnAppSend().

#define TCP_TIME_WAIT_TIME (2 * TCP_TICK_HZ)

Referenced by TcpConfigurePcb().

#define TCP_TIME_WAIT_TIME_MAX (60 * TCP_TICK_HZ)

Referenced by TcpConfigurePcb().

#define TCP_TIMER_2MSL 5
TIME_WAIT timer.

Referenced by TcpDeliverData(), and TcpInput().

#define TCP_TIMER_CONNECT 0
Connection establishment timer.

Referenced by TcpInput(), TcpOnAppConnect(), and TcpRexmitTimeout().

#define TCP_TIMER_FINWAIT2 4
FIN_WAIT_2 timer.

Referenced by TcpInput().

#define TCP_TIMER_KEEPALIVE 3
Keepalive timer.

Referenced by TcpSetKeepaliveTimer().

#define TCP_TIMER_NUMBER 6
The total number of the TCP timer.

Referenced by TcpTickingDpc(), and TcpUpdateTimer().

```
#define TCP_TIMER_ON ( Flag,
                       Timer    )    ((Flag) & (1 << (Timer)))
```

Referenced by TcpDataToSend(), TcpProbeTimeout(), TcpRexmitTimeout(), TcpSetKeepaliveTimer(), TcpTickingDpc(), TcpToSendData(), and TcpUpdateTimer().

#define TCP_TIMER_PROBE 2
Window probe timer.

Referenced by TcpInput(), and TcpSetProbeTimer().

#define TCP_TIMER_REXMIT 1
Retransmit timer.

Referenced by TcpDataToSend(), TcpInput(), TcpProbeTimeout(), TcpRexmitTimeout(), and TcpToSendData().

#define TCPSEG_NETBUF (NBuf) ((TCP_SEG *) ((NBuf)->ProtoData))

Referenced by TcpAdjustSndQue(), TcpBuildOption(), TcpCheckSndQue(), TcpDeliverData(), TcpFormatNetbuf(), TcpGetMaxSndNxt(), TcpGetSegmentSndQue(), TcpGetSegmentSock(), TcpQueueData(), TcpSendAck(), TcpSendZeroProbe(), TcpSynBuildOption(), TcpToSendData(), TcpTransmitSegment(), TcpTrimSegment(), and TcpVerifySegment().

Typedef Documentation

typedef struct _TCP_CONTROL_BLOCK TCP_CB

typedef struct _TCP_PEER TCP_PEER
Network endpoint, IP plus Port structure.

typedef struct _TCP_SEG TCP_SEG
TCP segmentation data.

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/TcpDxe/TcpTimer.c File Reference

Functions

VOID TcpConnectTimeout (IN OUT TCP_CB *Tcb)
VOID TcpRexmitTimeout (IN OUT TCP_CB *Tcb)
VOID TcpProbeTimeout (IN OUT TCP_CB *Tcb)
VOID TcpKeepaliveTimeout (IN OUT TCP_CB *Tcb)
VOID TcpFinwait2Timeout (IN OUT TCP_CB *Tcb)
VOID Tcp2MSLTimeout (IN OUT TCP_CB *Tcb)
VOID TcpClose (IN OUT TCP_CB *Tcb)
VOID TcpBackoffRto (IN OUT TCP_CB *Tcb)
VOID TcpUpdateTimer (IN OUT TCP_CB *Tcb)
VOID TcpSetTimer (IN OUT TCP_CB *Tcb, IN UINT16 Timer, IN UINT32 TimeOut)
VOID TcpClearTimer (IN OUT TCP_CB *Tcb, IN UINT16 Timer)
VOID TcpClearAllTimer (IN OUT TCP_CB *Tcb)
VOID TcpSetProbeTimer (IN OUT TCP_CB *Tcb)
VOID TcpSetKeepaliveTimer (IN OUT TCP_CB *Tcb)
VOID EFIAPI TcpTickingDpc (IN VOID *Context)
VOID EFIAPI TcpTicking (IN EFI_EVENT Event, IN VOID *Context)

Variables

UINT32 mTcpTick = 1000
TCP_TIMER_HANDLER mTcpTimerHandler [TCP_TIMER_NUMBER]

Detailed Description

TCP timer related functions.

Copyright (c) 2009 - 2010, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

VOID Tcp2MSLTimeout (IN OUT TCP_CB * *Tcb*)
Timeout handler for 2MSL timer.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References TcpClose().

VOID TcpBackoffRto (IN OUT TCP_CB * *Tcb*)

Backoff the RTO.

Parameters:

[in,out] *Tcb* Pointer to the TCP_CB of this TCP instance.

References TCP_FOLD_RTT, TCP_RTO_MAX, and TCP_RTO_MIN.

Referenced by TcpRexmitTimeout().

VOID TcpClearAllTimer (IN OUT TCP_CB * *Tcb*)

Clear all TCP timers.

Parameters:

[in,out] *Tcb* Pointer to the TCP_CB of this TCP instance.

References TcpUpdateTimer().

Referenced by TcpDeliverData(), and TcpInput().

VOID TcpClearTimer (IN OUT TCP_CB * *Tcb*,
IN UINT16 *Timer*
)

Clear one TCP timer.

Parameters:

[in,out] *Tcb* Pointer to the TCP_CB of this TCP instance.
[in] *Timer* The index of the timer to be cleared.

References TCP_CLEAR_TIMER, and TcpUpdateTimer().

Referenced by TcpInput().

VOID TcpClose (IN OUT TCP_CB * *Tcb*)

Close the TCP connection.

Parameters:

[in,out] *Tcb* Pointer to the TCP_CB of this TCP instance.

References TCP_CLOSED, and TcpSetState().

Referenced by Tcp2MSLTimeout(), TcpConnectTimeout(), TcpDeliverData(), TcpFinwait2Timeout(), TcpIcmpInput(), TcpInput(), TcpKeepaliveTimeout(), TcpOnAppClose(), and TcpRexmitTimeout().

VOID TcpConnectTimeout (IN OUT TCP_CB * *Tcb*)

Connect timeout handler.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References SOCK_ERROR, TCP_CONNECTED, TCP_SYN_RCVD, TcpClose(), and TcpResetConnection().

VOID TcpFinwait2Timeout (IN OUT TCP_CB * *Tcb*)

Timeout handler for FIN_WAIT_2 timer.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References TcpClose().

VOID TcpKeepaliveTimeout (IN OUT TCP_CB * *Tcb*)

Timeout handler for keepalive timer.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References SOCK_ERROR, TcpClose(), TcpSendZeroProbe(), and TcpSetKeepaliveTimer().

VOID TcpProbeTimeout (IN OUT TCP_CB * *Tcb*)

Timeout handler for window probe timer.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References TCP_TIMER_ON, TCP_TIMER_REXMIT, TcpDataToSend(), TcpSendZeroProbe(), TcpSetProbeTimer(), and TcpToSendData().

VOID TcpRexmitTimeout (IN OUT TCP_CB * *Tcb*)

Timeout handler for TCP retransmission timer.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References SOCK_ERROR, TCP_CLEAR_FLG, TCP_CONGEST_LOSS, TCP_CTRL_RTT_ON, TCP_SUB_SEQ, TCP_TIMER_CONNECT, TCP_TIMER_ON, TCP_TIMER_REXMIT, TcpBackoffRto(), TcpClose(), TcpRetransmit(), and TcpSetTimer().

VOID TcpSetKeepaliveTimer (IN OUT TCP_CB * *Tcb*)

Enable the keepalive timer and set the timeout value.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References TCP_CTRL_NO_KEEPALIVE, TCP_FLG_ON, TCP_TIMER_KEEPALIVE, TCP_TIMER_ON, and TcpSetTimer().

Referenced by TcpInput(), and TcpKeepaliveTimeout().

VOID TcpSetProbeTimer (IN OUT TCP_CB * *Tcb*)

Enable the window prober timer and set the timeout value.

Parameters:

`[in,out]` *Tcb* Pointer to the TCP_CB of this TCP instance.

References TCP_RTO_MAX, TCP_RTO_MIN, TCP_TIMER_PROBE, and TcpSetTimer().

Referenced by TcpDataToSend(), TcpInput(), and TcpProbeTimeout().

```
VOID TcpSetTimer ( IN OUT TCP_CB *   Tcb,
                   IN UINT16         Timer,
                   IN UINT32         TimeOut
                 )
```

Enable a TCP timer.

Parameters:

`[in,out]`	*Tcb*	Pointer to the TCP_CB of this TCP instance.
`[in]`	*Timer*	The index of the timer to be enabled.
`[in]`	*TimeOut*	The timeout value of this timer.

References mTcpTick, TCP_SET_TIMER, and TcpUpdateTimer().

Referenced by TcpDeliverData(), TcpInput(), TcpOnAppConnect(), TcpRexmitTimeout(), TcpSetKeepaliveTimer(), TcpSetProbeTimer(), and TcpToSendData().

```
VOID EFIAPI TcpTicking ( IN EFI_EVENT   Event,
                         IN VOID *      Context
                       )
```

Heart beat timer handler, queues the DPC at TPL_CALLBACK.

Parameters:

`[in]`	*Event*	Timer event signaled, ignored.
`[in]`	*Context*	Context of the timer event, ignored.

References TcpTickingDpc().

Referenced by TcpCreateTimer().

VOID EFIAPI TcpTickingDpc (IN VOID * *Context*)

Heart beat timer handler.

Parameters:

[in] *Context* Context of the timer event, ignored.

References _TCP_CONTROL_BLOCK::CtrlFlag, _TCP_CONTROL_BLOCK::DelayedAck, _TCP_CONTROL_BLOCK::EnabledTimer, _TCP_CONTROL_BLOCK::Idle, _TCP_CONTROL_BLOCK::IpInfo, mTcpGlobalIss, mTcpRunQue, mTcpTick, mTcpTimerHandler, _TCP_CONTROL_BLOCK::NextExpire, _TCP_CONTROL_BLOCK::RttMeasure, _TCP_CONTROL_BLOCK::State, TCP_CLEAR_TIMER, TCP_CLOSED, TCP_CTRL_RTT_ON, TCP_CTRL_TIMER_ON, TCP_FLG_ON, TCP_ISS_INCREMENT_2, TCP_TIME_LEQ, TCP_TIMER_NUMBER, TCP_TIMER_ON, TcpSendAck(), TcpUpdateTimer(), _TCP_CONTROL_BLOCK::Tick, and _TCP_CONTROL_BLOCK::Timer.

Referenced by TcpTicking().

VOID TcpUpdateTimer (IN OUT TCP_CB * *Tcb*)

Update the timer status and the next expire time according to the timers to expire in a specific future time slot.

Parameters:

[in,out] *Tcb* Pointer to the TCP_CB of this TCP instance.

References mTcpTick, TCP_CLEAR_FLG, TCP_CTRL_TIMER_ON, TCP_EXPIRE_TIME, TCP_SET_FLG, TCP_SUB_TIME, TCP_TIME_LT, TCP_TIMER_NUMBER, and TCP_TIMER_ON.

Referenced by TcpClearAllTimer(), TcpClearTimer(), TcpSetTimer(), and TcpTickingDpc().

Variable Documentation

UINT32 mTcpTick = 1000

Referenced by TcpBuildOption(), TcpInput(), TcpPawsOK(), TcpSetTimer(), TcpSynBuildOption(), TcpTickingDpc(), and TcpUpdateTimer().

TCP_TIMER_HANDLER mTcpTimerHandler[TCP_TIMER_NUMBER]

Initial value:

```
{
 TcpConnectTimeout,
 TcpRexmitTimeout,
 TcpProbeTimeout,
 TcpKeepaliveTimeout,
 TcpFinwait2Timeout,
 Tcp2MSLTimeout,
}
```

Referenced by TcpTickingDpc().

NetworkPkg/Udp6Dxe/ComponentName.c File Reference

Functions

EFI_STATUS EFIAPI Udp6ComponentNameGetDriverName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN CHAR8 *Language, OUT CHAR16 **DriverName)

EFI_STATUS EFIAPI Udp6ComponentNameGetControllerName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE ChildHandle, IN CHAR8 *Language, OUT CHAR16 **ControllerName)

EFI_STATUS UpdateName (IN EFI_UDP6_PROTOCOL *Udp6)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gUdp6ComponentName

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gUdp6ComponentName2

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mUdp6DriverNameTable []

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE * gUdp6ControllerNameTable = NULL

Detailed Description

UEFI Component Name(2) protocol implementation for UDP6 driver.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI Udp6ComponentNameGetControllerName (	IN EFI_COMPONENT_NAME_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_HANDLE	*ChildHandle*,
	IN CHAR8 *	*Language*,

OUT CHAR16 ** *ControllerName*
)

Retrieves a Unicode string that is the user-readable name of the controller that is being managed by a driver.

This function retrieves the user-readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

`[in]`	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
`[in]`	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
`[in]`	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
`[in]`	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
`[out]`	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

Retrieves a Unicode string that is the user-readable name of the controller that is being managed by a driver.

This function retrieves the user-readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
[in]	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL, and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gUdp6ComponentName, gUdp6ControllerNameTable, and UpdateName().

```
EFI_STATUS EFIAPI
Udp6ComponentNameGetDriverName
  (
    IN EFI_COMPONENT_NAME_PROTOCOL *   This,
    IN CHAR8 *                         Language,
    OUT CHAR16 **                      DriverName
  )
```

Retrieves a Unicode string that is the user-readable name of the driver.

This function retrieves the user-readable name of a driver in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the driver name is returned in DriverName, and EFI_SUCCESS is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.

[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*DriverName*	A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by This in the language specified by Language.

Return values:

EFI_SUCCESS	The Unicode string for the Driver specified by This and the language specified by Language was returned in DriverName.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	DriverName is NULL.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gUdp6ComponentName, and mUdp6DriverNameTable.

EFI_STATUS UpdateName (IN EFI_UDP6_PROTOCOL * *Udp6*)

Update the component name for the Udp6 child handle.

Parameters:

Udp6[in] A pointer to the EFI_UDP6_PROTOCOL.

Return values:

EFI_SUCCESS	Update the ControllerNameTable of this instance successfully.
EFI_INVALID_PARAMETER	The input parameter is invalid.

References gUdp6ComponentName, gUdp6ComponentName2, and gUdp6ControllerNameTable.

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gUdp6ComponentName

Initial value:

```
{
 Udp6ComponentNameGetDriverName,
 Udp6ComponentNameGetControllerName,
 "eng"
}
```

Referenced by Udp6ComponentNameGetControllerName(), Udp6ComponentNameGetDriverName(), Udp6DriverEntryPoint(), and UpdateName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gUdp6ComponentName2

Initial value:

```
{
 (EFI_COMPONENT_NAME2_GET_DRIVER_NAME) Udp6ComponentNameGetDriverName,
 (EFI_COMPONENT_NAME2_GET_CONTROLLER_NAME) Udp6ComponentNameGetControllerName,
 "en"
}
```

Referenced by Udp6DriverEntryPoint(), and UpdateName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE* gUdp6ControllerNameTable = NULL

Referenced by Udp6ComponentNameGetControllerName(), and UpdateName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mUdp6DriverNameTable[]

Initial value:

```
 {
  {
    "eng;en",
    L"UDP6 Network Service Driver"
  },
  {
    NULL,
    NULL
  }
}
```

Referenced by Udp6ComponentNameGetDriverName().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Udp6Dxe/Udp6Driver.c File Reference

Functions

EFI_STATUS EFIAPI Udp6DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI Udp6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI Udp6DestroyChildEntryInHandleBuffer (IN LIST_ENTRY *Entry, IN VOID *Context)
EFI_STATUS EFIAPI Udp6DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)
EFI_STATUS EFIAPI Udp6ServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN OUT EFI_HANDLE *ChildHandle)
EFI_STATUS EFIAPI Udp6ServiceBindingDestroyChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE ChildHandle)
EFI_STATUS EFIAPI Udp6DriverEntryPoint (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)

Variables

EFI_DRIVER_BINDING_PROTOCOL gUdp6DriverBinding
EFI_SERVICE_BINDING_PROTOCOL mUdp6ServiceBinding

Detailed Description

Driver Binding functions and Service Binding functions for the Network driver module.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI Udp6DestroyChildEntryInHandleBuffer (*Entry*,

```
                                    IN
                                    LIST_ENTRY
                                    *
                                    IN VOID *        Context
                                )
```

Callback function which provided by user to remove one node in NetDestroyLinkList process.

Parameters:

[in]	*Entry*	The entry to be removed.
[in]	*Context*	Pointer to the callback context corresponds to the Context in NetDestroyLinkList.

Return values:

EFI_SUCCESS	The entry has been removed successfully.
Others	Fail to remove the entry.

References _UDP6_INSTANCE_DATA::ChildHandle, and UDP6_INSTANCE_DATA_SIGNATURE.

Referenced by Udp6DriverBindingStop().

```
EFI_STATUS EFIAPI
Udp6DriverBindingStart    ( IN EFI_DRIVER_BINDING_PROTOCOL *   This,
                            IN EFI_HANDLE                      ControllerHandle,
                            IN EFI_DEVICE_PATH_PROTOCOL *      RemainingDevicePath
                          )
```

Start this driver on ControllerHandle.

This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Start() it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to bind the driver to.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCES	This driver is added to ControllerHandle.
EFI_OUT_OF_RESOURCES	The required system resource can't be allocated.
other	This driver does not support this device.

References _UDP6_SERVICE_DATA::ServiceBinding, Udp6CleanService(), and Udp6CreateService().

```
EFI_STATUS EFIAPI
Udp6DriverBindingStop     ( IN EFI_DRIVER_BINDING_PROTOCOL *   This,
                            IN EFI_HANDLE                      ControllerHandle,
                            IN UINTN                           NumberOfChildren,
                            IN EFI_HANDLE *                    ChildHandleBuffer
                          )
```

Stop this driver on ControllerHandle.

This service is called by the EFI boot service DisconnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. DisconnectController() must follow these calling restrictions. If any other agent wishes to call Stop(), it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to stop the driver on.
[in]	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If the number of children is zero stop the entire bus driver.
[in]	*ChildHandleBuffer*	List of Child Handles to Stop. It is optional.

Return values:

EFI_SUCCES	This driver is removed ControllerHandle.
EFI_DEVICE_ERROR	Can't find the NicHandle from the ControllerHandle and specified GUID.
other	This driver was not removed from this device.

References UDP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer, _UDP6_SERVICE_DATA::ChildrenList, UDP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren, _UDP6_SERVICE_DATA::ServiceBinding, UDP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding, UDP6_SERVICE_DATA_FROM_THIS, Udp6CleanService(), and Udp6DestroyChildEntryInHandleBuffer().

```
EFI_STATUS EFIAPI
Udp6DriverBindingSupported (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
)
```

Tests to see if this driver supports a given controller. If a child device is provided, it further tests to see if this driver supports creating a handle for the specified child device.

This function checks to see if the driver specified by This supports the device specified by ControllerHandle. Drivers will typically use the device path attached to ControllerHandle and/or the services from the bus I/O abstraction attached to ControllerHandle to determine if the driver supports ControllerHandle. This function may be called many times during platform initialization. In order to reduce boot times, the tests performed by this function must be very small, and take as little time as possible to execute. This function must not change the state of any hardware devices, and this function must be aware that the device specified by ControllerHandle may already be managed by the same driver or a different driver. This function must match its calls to AllocatePages() with FreePages(), AllocatePool() with FreePool(), and OpenProtocol() with CloseProtocol(). Because ControllerHandle may have been previously started by the same driver, if a protocol is already in the opened state, then it must not be closed with CloseProtocol(). This is required to guarantee the state of ControllerHandle is not modified by this function.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the

child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by the driver specified by This.
EFI_ACCESS_DENIED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by a different driver or an application that requires exclusive access. Currently not implemented.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

```
EFI_STATUS EFIAPI Udp6DriverEntryPoint ( IN EFI_HANDLE          ImageHandle,
                                         IN EFI_SYSTEM_TABLE *  SystemTable
                                       )
```

This is the declaration of an EFI image entry point. This entry point is the same for UEFI Applications, UEFI OS Loaders, and UEFI Drivers, including both device drivers and bus drivers.

The entry point for Udp6 driver that installs the driver binding and component name protocol on its ImageHandle.

Parameters:

`[in]`	*ImageHandle*	The firmware allocated handle for the UEFI image.
`[in]`	*SystemTable*	A pointer to the EFI System Table.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_OUT_OF_RESOURCES	The request could not be completed due to a lack of resources.

References gUdp6ComponentName, gUdp6ComponentName2, gUdp6DriverBinding, mUdp6RandomPort, and UDP6_PORT_KNOWN.

```
EFI_STATUS EFIAPI                    ( IN EFI_SERVICE_BINDING_PROTOCOL *  This,
Udp6ServiceBindingCreateChild          IN OUT EFI_HANDLE *                ChildHandle
                                     )
```

Creates a child handle and installs a protocol.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

`[in]`	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
`[in,out]`	*ChildHandle*	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.
EFI_INVALID_PARAMETER	This is NULL or ChildHandle is NULL.
EFI_OUT_OF_RESOURCES	There are not enough resources availabe to create the child.
other	The child handle was not created.

References _UDP6_INSTANCE_DATA::ChildHandle, _UDP6_SERVICE_DATA::ChildrenList, _UDP6_SERVICE_DATA::ChildrenNumber, gUdp6DriverBinding, _UDP6_INSTANCE_DATA::IpInfo, _UDP6_SERVICE_DATA::IpIo, _UDP6_INSTANCE_DATA::Link, UDP6_SERVICE_DATA_FROM_THIS, Udp6CleanInstance(), Udp6InitInstance(), and _UDP6_INSTANCE_DATA::Udp6Proto.

EFI_STATUS EFIAPI Udp6ServiceBindingDestroyChild	(IN EFI_SERVICE_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ChildHandle*
	)	

Destroys a child handle with a set of I/O services. The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ChildHandle*	Handle of the child to destroy.

Return values:

EFI_SUCCES	The I/O services were removed from the child handle.
EFI_UNSUPPORTED	The child handle does not support the I/O services that are being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
EFI_ACCESS_DENIED	The child handle could not be destroyed because its I/O services are being used.
other	The child handle was not destroyed.

References _UDP6_INSTANCE_DATA::ChildHandle, _UDP6_SERVICE_DATA::ChildrenNumber, gUdp6DriverBinding, _UDP6_INSTANCE_DATA::InDestroy, _UDP6_INSTANCE_DATA::IpInfo, _UDP6_SERVICE_DATA::IpIo, _UDP6_INSTANCE_DATA::Link, UDP6_INSTANCE_DATA_FROM_THIS, UDP6_SERVICE_DATA_FROM_THIS, Udp6CleanInstance(), and _UDP6_INSTANCE_DATA::Udp6Proto.

Variable Documentation

EFI_DRIVER_BINDING_PROTOCOL gUdp6DriverBinding

Initial value:

```
{
 Udp6DriverBindingSupported,
 Udp6DriverBindingStart,
 Udp6DriverBindingStop,
 0xa,
 NULL,
 NULL
}
```

Referenced by Udp6DriverEntryPoint(), Udp6ServiceBindingCreateChild(), and Udp6ServiceBindingDestroyChild().

EFI_SERVICE_BINDING_PROTOCOL mUdp6ServiceBinding

Initial value:

```
{
  Udp6ServiceBindingCreateChild,
  Udp6ServiceBindingDestroyChild
}
```

Referenced by Udp6CreateService().

Generated on Mon Sep 28 08:49:56 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Udp6Dxe/Udp6Driver.h File Reference

Functions

EFI_STATUS EFIAPI	Udp6DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI	Udp6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI	Udp6DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)
EFI_STATUS EFIAPI	Udp6ServiceBindingCreateChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN OUT EFI_HANDLE *ChildHandle)
EFI_STATUS EFIAPI	Udp6ServiceBindingDestroyChild (IN EFI_SERVICE_BINDING_PROTOCOL *This, IN EFI_HANDLE ChildHandle)

Detailed Description

Driver Binding functions and Service Binding functions for the Network driver module.

Copyright (c) 2009 - 2011, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI Udp6DriverBindingStart	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*
	)	

Start this driver on ControllerHandle.

This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Start() it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to bind a driver to.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCES	This driver is added to ControllerHandle.
EFI_OUT_OF_RESOURCES	The required system resource can't be allocated.
other	This driver does not support this device.

Start this driver on ControllerHandle.

This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Start() it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to bind the driver to.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to start.

Return values:

EFI_SUCCES	This driver is added to ControllerHandle.
EFI_OUT_OF_RESOURCES	The required system resource can't be allocated.
other	This driver does not support this device.

References _UDP6_SERVICE_DATA::ServiceBinding, Udp6CleanService(), and Udp6CreateService().

```
EFI_STATUS EFIAPI
Udp6DriverBindingStop
(
  IN EFI_DRIVER_BINDING_PROTOCOL *   This,
  IN EFI_HANDLE                      ControllerHandle,
  IN UINTN                           NumberOfChildren,
  IN EFI_HANDLE *                    ChildHandleBuffer
)
```

Stop this driver on ControllerHandle.

This service is called by the EFI boot service DisconnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. DisconnectController() must follow these calling restrictions. If any other agent wishes to call Stop(), it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to stop the driver on.
[in]	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If number of children is zero, stop the entire bus driver.
[in]	*ChildHandleBuffer*	List of Child Handles to Stop. It is optional.

Return values:

EFI_SUCCESS	This driver removed ControllerHandle.
EFI_DEVICE_ERROR	Can't find the NicHandle from the ControllerHandle and specified GUID.
other	This driver was not removed from this device.

Stop this driver on ControllerHandle.

This service is called by the EFI boot service DisconnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. DisconnectController() must follow these calling

restrictions. If any other agent wishes to call Stop(), it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to stop the driver on.
[in]	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If the number of children is zero stop the entire bus driver.
[in]	*ChildHandleBuffer*	List of Child Handles to Stop. It is optional.

Return values:

EFI_SUCCES	This driver is removed ControllerHandle.
EFI_DEVICE_ERROR	Can't find the NicHandle from the ControllerHandle and specified GUID.
other	This driver was not removed from this device.

References UDP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ChildHandleBuffer, _UDP6_SERVICE_DATA::ChildrenList, UDP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::NumberOfChildren, _UDP6_SERVICE_DATA::ServiceBinding, UDP6_DESTROY_CHILD_IN_HANDLE_BUF_CONTEXT::ServiceBinding, UDP6_SERVICE_DATA_FROM_THIS, Udp6CleanService(), and Udp6DestroyChildEntryInHandleBuffer().

EFI_STATUS EFIAPI Udp6DriverBindingSupported	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*
	)	

Tests to see if this driver supports a given controller. If a child device is provided, it further tests to see if this driver supports creating a handle for the specified child device.

This function checks to see if the driver specified by This supports the device specified by ControllerHandle. Drivers typically use the device path attached to ControllerHandle and/or the services from the bus I/O abstraction attached to ControllerHandle to determine if the driver supports ControllerHandle. This function may be called many times during platform initialization. In order to reduce boot times, the tests performed by this function must be very small, and take as little time as possible to execute. This function must not change the state of any hardware devices, and this function must be aware that the device specified by ControllerHandle may already be managed by the same driver or a different driver. This function must match its calls to AllocatePages() with FreePages(), AllocatePool() with FreePool(), and OpenProtocol() with CloseProtocol(). Since ControllerHandle may have been previously started by the same driver, if a protocol is already in the opened state, then it must not be closed with CloseProtocol(). This is required to guarantee the state of ControllerHandle is not modified by this function.

Parameters:

[in]	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
[in]	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by the driver specified by This.
EFI_ACCESS_DENIED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by a different driver or an application that requires exclusive access. Currently not implemented.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

Tests to see if this driver supports a given controller. If a child device is provided, it further tests to see if this driver supports creating a handle for the specified child device.

This function checks to see if the driver specified by This supports the device specified by ControllerHandle. Drivers will typically use the device path attached to ControllerHandle and/or the services from the bus I/O abstraction attached to ControllerHandle to determine if the driver supports ControllerHandle. This function may be called many times during platform initialization. In order to reduce boot times, the tests performed by this function must be very small, and take as little time as possible to execute. This function must not change the state of any hardware devices, and this function must be aware that the device specified by ControllerHandle may already be managed by the same driver or a different driver. This function must match its calls to AllocatePages() with FreePages(), AllocatePool() with FreePool(), and OpenProtocol() with CloseProtocol(). Because ControllerHandle may have been previously started by the same driver, if a protocol is already in the opened state, then it must not be closed with CloseProtocol(). This is required to guarantee the state of ControllerHandle is not modified by this function.

Parameters:

`[in]`	*This*	A pointer to the EFI_DRIVER_BINDING_PROTOCOL instance.
`[in]`	*ControllerHandle*	The handle of the controller to test. This handle must support a protocol interface that supplies an I/O abstraction to the driver.
`[in]`	*RemainingDevicePath*	A pointer to the remaining portion of a device path. This parameter is ignored by device drivers, and is optional for bus drivers. For bus drivers, if this parameter is not NULL, then the bus driver must determine if the bus controller specified by ControllerHandle and the child controller specified by RemainingDevicePath are both supported by this bus driver.

Return values:

EFI_SUCCESS	The device specified by ControllerHandle and RemainingDevicePath is supported by the driver specified by This.
EFI_ALREADY_STARTED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by the driver specified by This.
EFI_ACCESS_DENIED	The device specified by ControllerHandle and RemainingDevicePath is already being managed by a different driver or an application that requires exclusive access. Currently not implemented.
EFI_UNSUPPORTED	The device specified by ControllerHandle and RemainingDevicePath is not supported by the driver specified by This.

```
EFI_STATUS EFIAPI
Udp6ServiceBindingCreateChild
(
  IN EFI_SERVICE_BINDING_PROTOCOL *   This,
  IN OUT EFI_HANDLE *                 ChildHandle
)
```

Creates a child handle and installs a protocol.

The CreateChild() function installs a protocol on ChildHandle. If ChildHandle is a pointer to NULL, then a new handle is created and returned in ChildHandle. If ChildHandle is not a pointer to NULL, then the protocol installs on the existing ChildHandle.

Parameters:

`[in]`	*This*	Pointer to the EFI_SERVICE_BINDING_PROTOCOL instance.
`[in,out]`	*ChildHandle*	Pointer to the handle of the child to create. If it is NULL, then a new handle is created. If it is a pointer to an existing UEFI handle, then the protocol is added to the existing UEFI handle.

Return values:

EFI_SUCCES	The protocol was added to ChildHandle.
EFI_INVALID_PARAMETER	This is NULL or ChildHandle is NULL.
EFI_OUT_OF_RESOURCES	There are not enough resources availabe to create the child.
other	The child handle was not created.

References _UDP6_INSTANCE_DATA::ChildHandle, _UDP6_SERVICE_DATA::ChildrenList, _UDP6_SERVICE_DATA::ChildrenNumber, gUdp6DriverBinding, _UDP6_INSTANCE_DATA::IpInfo, _UDP6_SERVICE_DATA::IpIo, _UDP6_INSTANCE_DATA::Link, UDP6_SERVICE_DATA_FROM_THIS, Udp6CleanInstance(), Udp6InitInstance(), and _UDP6_INSTANCE_DATA::Udp6Proto.

```
EFI_STATUS EFIAPI
Udp6ServiceBindingDestroyChild
(
  IN EFI_SERVICE_BINDING_PROTOCOL *   This,
  IN EFI_HANDLE                       ChildHandle
)
```

Destroys a child handle with a set of I/O services. The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

`[in]`	*This*	Protocol instance pointer.
`[in]`	*ChildHandle*	Handle of the child to destroy.

Return values:

EFI_SUCCES	The I/O services were removed from the child handle.
EFI_UNSUPPORTED	The child handle does not support the I/O services that are being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
EFI_ACCESS_DENIED	The child handle could not be destroyed because its I/O services are being used.
other	The child handle was not destroyed.

Destroys a child handle with a set of I/O services. The DestroyChild() function does the opposite of CreateChild(). It removes a protocol that was installed by CreateChild() from ChildHandle. If the removed protocol is the last protocol on ChildHandle, then ChildHandle is destroyed.

Parameters:

`[in]`	*This*	Protocol instance pointer.
`[in]`	*ChildHandle*	Handle of the child to destroy.

Return values:

EFI_SUCCES	The I/O services were removed from the child handle.
EFI_UNSUPPORTED	The child handle does not support the I/O services that are being removed.
EFI_INVALID_PARAMETER	Child handle is NULL.
EFI_ACCESS_DENIED	The child handle could not be destroyed because its I/O services are being used.
other	The child handle was not destroyed.

References _UDP6_INSTANCE_DATA::ChildHandle, _UDP6_SERVICE_DATA::ChildrenNumber, gUdp6DriverBinding, _UDP6_INSTANCE_DATA::InDestroy, _UDP6_INSTANCE_DATA::IpInfo, _UDP6_SERVICE_DATA::IpIo, _UDP6_INSTANCE_DATA::Link, UDP6_INSTANCE_DATA_FROM_THIS, UDP6_SERVICE_DATA_FROM_THIS, Udp6CleanInstance(), and _UDP6_INSTANCE_DATA::Udp6Proto.

Generated on Mon Sep 28 08:49:56 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Udp6Dxe/Udp6Impl.c File Reference

Functions

VOID EFIAPI Udp6CheckTimeout (IN EFI_EVENT Event, IN VOID *Context)
BOOLEAN Udp6FindInstanceByPort (IN LIST_ENTRY *InstanceList, IN EFI_IPv6_ADDRESS *Address, IN UINT16 Port)
VOID EFIAPI Udp6DgramSent (IN EFI_STATUS Status, IN VOID *Context, IN IP_IO_IP_PROTOCOL Sender, IN VOID *NotifyData)
VOID EFIAPI Udp6DgramRcvd (IN EFI_STATUS Status, IN UINT8 IcmpError, IN EFI_NET_SESSION_DATA *NetSession, IN NET_BUF *Packet, IN VOID *Context)
EFI_STATUS EFIAPI Udp6CancelTokens (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Arg)
BOOLEAN Udp6MatchDgram (IN UDP6_INSTANCE_DATA *Instance, IN EFI_UDP6_SESSION_DATA *Udp6Session)
VOID EFIAPI Udp6RecycleRxDataWrap (IN EFI_EVENT Event, IN VOID *Context)
UDP6_RXDATA_WRAP * Udp6WrapRxData (IN UDP6_INSTANCE_DATA *Instance, IN NET_BUF *Packet, IN EFI_UDP6_RECEIVE_DATA *RxData)
UINTN Udp6EnqueueDgram (IN UDP6_SERVICE_DATA *Udp6Service, IN NET_BUF *Packet, IN EFI_UDP6_RECEIVE_DATA *RxData)
VOID Udp6DeliverDgram (IN UDP6_SERVICE_DATA *Udp6Service)
VOID Udp6Demultiplex (IN UDP6_SERVICE_DATA *Udp6Service, IN EFI_NET_SESSION_DATA *NetSession, IN NET_BUF *Packet)
VOID Udp6IcmpHandler (IN UDP6_SERVICE_DATA *Udp6Service, IN UINT8 IcmpError, IN EFI_NET_SESSION_DATA *NetSession, IN OUT NET_BUF *Packet)
VOID Udp6SendPortUnreach (IN IP_IO *IpIo, IN EFI_NET_SESSION_DATA *NetSession, IN VOID *Udp6Header)
NET_MAP_ITEM * Udp6MapMultiCastAddr (IN NET_MAP *Map, IN VOID *Key)
EFI_STATUS Udp6CreateService (IN UDP6_SERVICE_DATA *Udp6Service, IN EFI_HANDLE ImageHandle, IN EFI_HANDLE ControllerHandle)
VOID Udp6CleanService (IN OUT UDP6_SERVICE_DATA *Udp6Service)
VOID Udp6InitInstance (IN UDP6_SERVICE_DATA *Udp6Service, IN OUT UDP6_INSTANCE_DATA *Instance)
VOID Udp6CleanInstance (IN OUT UDP6_INSTANCE_DATA *Instance)
EFI_STATUS Udp6Bind (IN LIST_ENTRY *InstanceList, IN EFI_UDP6_CONFIG_DATA *ConfigData)
BOOLEAN Udp6IsReconfigurable (IN EFI_UDP6_CONFIG_DATA *OldConfigData, IN EFI_UDP6_CONFIG_DATA *NewConfigData)
VOID Udp6BuildIp6ConfigData (IN EFI_UDP6_CONFIG_DATA *Udp6ConfigData, IN OUT EFI_IP6_CONFIG_DATA *Ip6ConfigData)
EFI_STATUS Udp6ValidateTxToken (IN UDP6_INSTANCE_DATA *Instance, IN EFI_UDP6_COMPLETION_TOKEN *TxToken)
EFI_STATUS EFIAPI Udp6TokenExist (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Context)
UINT16 Udp6Checksum (IN NET_BUF *Packet, IN UINT16 HeadSum)
EFI_STATUS Udp6RemoveToken (IN NET_MAP *TokenMap, IN EFI_UDP6_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Udp6LeaveGroup (IN NET_MAP *Map, IN NET_MAP_ITEM *Item, IN VOID *Arg)
VOID Udp6FlushRcvdDgram (IN UDP6_INSTANCE_DATA *Instance)

EFI_STATUS Udp6InstanceCancelToken (IN UDP6_INSTANCE_DATA *Instance, IN EFI_UDP6_COMPLETION_TOKEN *Token)
VOID Udp6InstanceDeliverDgram (IN UDP6_INSTANCE_DATA *Instance)
VOID Udp6ReportIcmpError (IN UDP6_INSTANCE_DATA *Instance)
VOID EFIAPI Udp6NetVectorExtFree (IN VOID *Context)

Variables

UINT16 mUdp6RandomPort

Detailed Description

Udp6 driver's whole implementation.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS Udp6Bind (IN LIST_ENTRY * *InstanceList*,
IN EFI_UDP6_CONFIG_DATA * *ConfigData*
)

This function tries to bind the udp instance according to the configured port allocation stragety.

Parameters:

[in]	*InstanceList*	Pointer to the head of the list linking the udp instances.
[in]	*ConfigData*	Pointer to the ConfigData of the instance to be bound.

Return values:

EFI_SUCCESS	The bound operation completed successfully.
EFI_ACCESS_DENIED	The <Address, Port> specified by the ConfigData is already used by other instance.
EFI_OUT_OF_RESOURCES	No available port resources.

References mUdp6RandomPort, UDP6_PORT_KNOWN, and Udp6FindInstanceByPort().

Referenced by Udp6Configure().

VOID Udp6BuildIp6ConfigData (IN EFI_UDP6_CONFIG_DATA * *Udp6ConfigData*,
IN OUT EFI_IP6_CONFIG_DATA * *Ip6ConfigData*
)

This function builds the Ip6 configdata from the Udp6ConfigData.

Parameters:

[in]	*Udp6ConfigData*	Pointer to the EFI_UDP6_CONFIG_DATA.
[in,out]	*Ip6ConfigData*	Pointer to the EFI_IP6_CONFIG_DATA.

Referenced by Udp6Configure().

```
EFI_STATUS EFIAPI Udp6CancelTokens ( IN NET_MAP *          Map,
                                     IN NET_MAP_ITEM *     Item,
                                     IN VOID *             Arg
                                   )
```

This function cancle the token specified by Arg in the Map.

Parameters:

[in]	*Map*	Pointer to the NET_MAP.
[in]	*Item*	Pointer to the NET_MAP_ITEM.
[in]	*Arg*	Pointer to the token to be cancelled, if NULL, all the tokens in this Map will be cancelled. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The token is cancelled if Arg is NULL or the token is not the same as that in the Item if Arg is not NULL.
EFI_ABORTED	Arg is not NULL, and the token specified by Arg is cancelled.

This function cancle the token specified by Arg in the Map.

Parameters:

[in]	*Map*	Pointer to the NET_MAP.
[in]	*Item*	Pointer to the NET_MAP_ITEM.
[in]	*Arg*	Pointer to the token to be cancelled. If NULL, all the tokens in this Map will be cancelled. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The token is cancelled if Arg is NULL, or the token is not the same as that in the Item, if Arg is not NULL.
EFI_ABORTED	Arg is not NULL, and the token specified by Arg is cancelled.

Referenced by Udp6InstanceCancelToken().

```
UINT16 Udp6Checksum ( IN NET_BUF *   Packet,
                      IN UINT16      HeadSum
                    )
```

This function calculates the checksum for the Packet, utilizing the pre-calculated pseudo HeadSum to reduce some overhead.

Parameters:

[in]	*Packet*	Pointer to the NET_BUF contains the udp datagram.
[in]	*HeadSum*	Checksum of the pseudo header, execpt the length field.

Returns:

The 16-bit checksum of this udp datagram.

Referenced by Udp6Demultiplex(), and Udp6Transmit().

VOID EFIAPI Udp6CheckTimeout (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

This function checks and timeouts the I/O datagrams holding by the corresponding service context.

Parameters:

[in] *Event* The event this function is registered to.
[in] *Context* The context data registered during the creation of the Event.

This function checks and times out the I/O datagrams listed in the UDP6_SERVICE_DATA which is specified by the input parameter Context.

Parameters:

[in] *Event* The event this function registered to.
[in] *Context* The context data registered during the creation of the Event.

References _UDP6_SERVICE_DATA::ChildrenList, _UDP6_INSTANCE_DATA::ConfigData, _UDP6_INSTANCE_DATA::Configured, _UDP6_INSTANCE_DATA::RcvdDgramQue, _UDP6_RXDATA_WRAP::TimeoutTick, UDP6_INSTANCE_DATA_SIGNATURE, UDP6_SERVICE_DATA_SIGNATURE, UDP6_TIMEOUT_INTERVAL, and Udp6RecycleRxDataWrap().

Referenced by Udp6CreateService().

VOID Udp6CleanInstance (IN OUT UDP6_INSTANCE_DATA * *Instance*)

This function cleans the udp instance.

Parameters:

[in,out] *Instance* Pointer to the UDP6_INSTANCE_DATA to clean.

Referenced by Udp6ServiceBindingCreateChild(), and Udp6ServiceBindingDestroyChild().

VOID Udp6CleanService (IN OUT UDP6_SERVICE_DATA * *Udp6Service*)

Clean the Udp service context data.

Parameters:

[in,out] *Udp6Service* Pointer to the UDP6_SERVICE_DATA.

Referenced by Udp6DriverBindingStart(), and Udp6DriverBindingStop().

EFI_STATUS Udp6CreateService (IN UDP6_SERVICE_DATA * *Udp6Service*,
IN EFI_HANDLE *ImageHandle*,
IN EFI_HANDLE *ControllerHandle*
)

Create the Udp service context data.

Parameters:

[in]	*Udp6Service*	Pointer to the UDP6_SERVICE_DATA.
[in]	*ImageHandle*	The image handle of this udp6 driver.
[in]	*ControllerHandle*	The controller handle this udp6 driver binds on.

Return values:

EFI_SUCCESS	The udp6 service context data was created and initialized.
EFI_OUT_OF_RESOURCES	Cannot allocate memory.
Others	An error condition occurred.

References mUdp6ServiceBinding, UDP6_SERVICE_DATA_SIGNATURE, UDP6_TIMEOUT_INTERVAL, Udp6CheckTimeout(), Udp6DgramRcvd(), and Udp6DgramSent().

Referenced by Udp6DriverBindingStart().

VOID Udp6DeliverDgram (IN UDP6_SERVICE_DATA * *Udp6Service*)

This function delivers the datagrams enqueued in the instances.

Parameters:

[in]	*Udp6Service*	Pointer to the udp service context data.

References _UDP6_INSTANCE_DATA::Configured, and Udp6InstanceDeliverDgram().

Referenced by Udp6Demultiplex().

VOID Udp6Demultiplex (IN UDP6_SERVICE_DATA * *Udp6Service*,
IN EFI_NET_SESSION_DATA * *NetSession*,
IN NET_BUF * *Packet*
)

This function demultiplexes the received udp datagram to the apropriate instances.

Parameters:

[in]	*Udp6Service*	Pointer to the udp service context data.
[in]	*NetSession*	Pointer to the EFI_NET_SESSION_DATA abstrated from the received datagram.
[in]	*Packet*	Pointer to the buffer containing the received udp datagram.

This function demultiplexes the received udp datagram to the appropriate instances.

Parameters:

[in]	*Udp6Service*	Pointer to the udp service context data.
[in]	*NetSession*	Pointer to the EFI_NET_SESSION_DATA abstrated from the received datagram.
[in]	*Packet*	Pointer to the buffer containing the received udp datagram.

References UDP6_HEADER_SIZE, Udp6Checksum(), Udp6DeliverDgram(), Udp6EnqueueDgram(), and Udp6SendPortUnreach().

Referenced by Udp6DgramRcvd().

```
VOID EFIAPI Udp6DgramRcvd ( IN EFI_STATUS                Status,
                            IN UINT8                     IcmpError,
                            IN EFI_NET_SESSION_DATA *    NetSession,
                            IN NET_BUF *                 Packet,
                            IN VOID *                    Context
                          )
```

This function processes the received datagram passed up by the IpIo layer.

Parameters:

[in]	*Status*	The status of this udp datagram.
[in]	*IcmpError*	The IcmpError code, only available when Status is EFI_ICMP_ERROR.
[in]	*NetSession*	Pointer to the EFI_NET_SESSION_DATA.
[in]	*Packet*	Pointer to the NET_BUF containing the received udp datagram.
[in]	*Context*	Pointer to the context data.

References Udp6Demultiplex(), and Udp6IcmpHandler().

Referenced by Udp6CreateService().

```
VOID EFIAPI Udp6DgramSent ( IN EFI_STATUS          Status,
                            IN VOID *              Context,
                            IN
                            IP_IO_IP_PROTOCOL      Sender,
                            IN VOID *              NotifyData
                          )
```

This function is the packet transmitting notify function registered to the IpIo interface. It's called to signal the udp TxToken when the IpIo layer completes transmitting of the udp datagram.

Parameters:

[in]	*Status*	The completion status of the output udp datagram.
[in]	*Context*	Pointer to the context data.
[in]	*Sender*	Specify a EFI_IP6_PROTOCOL for sending.
[in]	*NotifyData*	Pointer to the notify data.

This function is the packet transmitting notify function registered to the IpIo interface. It's called to signal the udp TxToken when IpIo layer completes the transmitting of the udp datagram.

Parameters:

[in]	*Status*	The completion status of the output udp datagram.
[in]	*Context*	Pointer to the context data.
[in]	*Sender*	Specify a EFI_IP6_PROTOCOL for sending.
[in]	*NotifyData*	Pointer to the notify data.

References _UDP6_INSTANCE_DATA::TxTokens, and Udp6RemoveToken().

Referenced by Udp6CreateService().

```
UINTN Udp6EnqueueDgram ( IN UDP6_SERVICE_DATA *      Udp6Service,
                         IN NET_BUF *                Packet,
```

IN EFI_UDP6_RECEIVE_DATA * *RxData*
)

This function enqueues the received datagram into the instances' receiving queues.

Parameters:

[in]	*Udp6Service*	Pointer to the udp service context data.
[in]	*Packet*	Pointer to the buffer containing the received datagram.
[in]	*RxData*	Pointer to the EFI_UDP6_RECEIVE_DATA of this datagram.

Returns:

The times this datagram is enqueued.

References _UDP6_INSTANCE_DATA::Configured, _UDP6_RXDATA_WRAP::Link, _UDP6_INSTANCE_DATA::RcvdDgramQue, Udp6MatchDgram(), and Udp6WrapRxData().

Referenced by Udp6Demultiplex().

BOOLEAN Udp6FindInstanceByPort (IN LIST_ENTRY * *InstanceList*,
IN EFI_IPv6_ADDRESS * *Address*,
IN UINT16 *Port*
)

This function finds the udp instance by the specified <Address, Port> pair.

Parameters:

[in]	*InstanceList*	Pointer to the head of the list linking the udp instances.
[in]	*Address*	Pointer to the specified IPv6 address.
[in]	*Port*	The udp port number.

Return values:

TRUE	The specified <Address, Port> pair is found.
FALSE	Otherwise.

References _UDP6_INSTANCE_DATA::ConfigData, and _UDP6_INSTANCE_DATA::Configured.

Referenced by Udp6Bind().

VOID Udp6FlushRcvdDgram (IN UDP6_INSTANCE_DATA * *Instance*)

This function removes all the Wrap datas in the RcvdDgramQue.

Parameters:

[in]	*Instance*	Pointer to the Udp6 Instance.

References Udp6RecycleRxDataWrap().

Referenced by Udp6Configure().

VOID Udp6IcmpHandler (IN UDP6_SERVICE_DATA * *Udp6Service*,
IN UINT8 *IcmpError*,
IN EFI_NET_SESSION_DATA * *NetSession*,

IN OUT NET_BUF * *Packet*
)

This function handles the received Icmp Error message and demultiplexes it to the instance.

Parameters:

[in]	*Udp6Service*	Pointer to the udp service context data.
[in]	*IcmpError*	The icmp error code.
[in]	*NetSession*	Pointer to the EFI_NET_SESSION_DATA abstracted from the received Icmp Error packet.
[in,out]	*Packet*	Pointer to the Icmp Error packet.

This function handles the received Icmp Error message and de-multiplexes it to the instance.

Parameters:

[in]	*Udp6Service*	Pointer to the udp service context data.
[in]	*IcmpError*	The icmp error code.
[in]	*NetSession*	Pointer to the EFI_NET_SESSION_DATA abstracted from the received Icmp Error packet.
[in,out]	*Packet*	Pointer to the Icmp Error packet.

References _UDP6_INSTANCE_DATA::Configured, _UDP6_INSTANCE_DATA::IcmpError, Udp6MatchDgram(), and Udp6ReportIcmpError().

Referenced by Udp6DgramRcvd().

VOID Udp6InitInstance (IN UDP6_SERVICE_DATA * *Udp6Service*,
IN OUT UDP6_INSTANCE_DATA * *Instance*
)

This function intializes the new created udp instance.

Parameters:

[in]	*Udp6Service*	Pointer to the UDP6_SERVICE_DATA.
[in,out]	*Instance*	Pointer to the un-initialized UDP6_INSTANCE_DATA.

References mUdp6Protocol, and UDP6_INSTANCE_DATA_SIGNATURE.

Referenced by Udp6ServiceBindingCreateChild().

EFI_STATUS Udp6InstanceCancelToken (IN UDP6_INSTANCE_DATA * *Instance*,
IN EFI_UDP6_COMPLETION_TOKEN * *Token*
)

Cancel Udp6 tokens from the Udp6 instance.

Parameters:

[in]	*Instance*	Pointer to the udp instance context data.
[in]	*Token*	Pointer to the token to be canceled. If NULL, all tokens in this instance will be cancelled. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The Token is cancelled.

EFI_NOT_FOUND The Token is not found.

References Udp6CancelTokens().

Referenced by Udp6Cancel().

VOID Udp6InstanceDeliverDgram (IN UDP6_INSTANCE_DATA * *Instance*)

This function delivers the received datagrams to the specified instance.

Parameters:

[in] *Instance* Pointer to the instance context data.

References _UDP6_RXDATA_WRAP::Link, _UDP6_RXDATA_WRAP::Packet, and _UDP6_RXDATA_WRAP::RxData.

Referenced by Udp6DeliverDgram(), and Udp6Receive().

BOOLEAN Udp6IsReconfigurable (IN EFI_UDP6_CONFIG_DATA * *OldConfigData*,
IN EFI_UDP6_CONFIG_DATA * *NewConfigData*
)

This function is used to check whether the NewConfigData has any un-reconfigurable parameters changed compared to the OldConfigData.

Parameters:

[in] *OldConfigData* Pointer to the current ConfigData the udp instance uses.

[in] *NewConfigData* Pointer to the new ConfigData.

Return values:

TRUE The instance is reconfigurable according to the NewConfigData.

FALSE Otherwise.

Referenced by Udp6Configure().

EFI_STATUS EFIAPI Udp6LeaveGroup (IN NET_MAP * *Map*,
IN NET_MAP_ITEM * *Item*,
IN VOID * *Arg*
)

This function removes the multicast group specified by Arg from the Map.

Parameters:

[in] *Map* Pointer to the NET_MAP.

[in] *Item* Pointer to the NET_MAP_ITEM.

[in] *Arg* Pointer to the Arg, it's the pointer to a multicast IPv6 Address. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS The multicast address is removed.

EFI_ABORTED The specified multicast address is removed, and the Arg is not NULL.

Referenced by Udp6Groups().

```
NET_MAP_ITEM * Udp6MapMultiCastAddr ( IN NET_MAP *  Map,
                                      IN VOID *     Key
                                    )
```

Find the key in the netmap

Parameters:

[in] *Map* The netmap to search within.
[in] *Key* The key to search.

Returns:

The point to the item contains the Key, or NULL if Key isn't in the map.

Find the key in the netmap.

Parameters:

[in] *Map* The netmap to search within.
[in] *Key* The key to search.

Returns:

The point to the item contains the Key, or NULL, if Key isn't in the map.

Referenced by Udp6MatchDgram().

```
BOOLEAN Udp6MatchDgram ( IN UDP6_INSTANCE_DATA *     Instance,
                         IN EFI_UDP6_SESSION_DATA *  Udp6Session
                       )
```

This function check if the received udp datagram matches with the Instance.

Parameters:

[in] *Instance* Pointer to the udp instance context data.
[in] *Udp6Session* Pointer to the EFI_UDP6_SESSION_DATA abstracted from the received udp datagram.

Return values:

TRUE The udp datagram matches the receiving requirements of the Instance.
FALSE The udp datagram doe not match the receiving requirements of the Instance.

This function checks if the received udp datagram matches with the Instance.

Parameters:

[in] *Instance* Pointer to the udp instance context data.
[in] *Udp6Session* Pointer to the EFI_UDP6_SESSION_DATA abstracted from the received udp datagram.

Return values:

TRUE The udp datagram matches the receiving requirements of the Instance.
FALSE The udp datagram does not matche the receiving requirements of the Instance.

References Udp6MapMultiCastAddr().

Referenced by Udp6EnqueueDgram(), and Udp6IcmpHandler().

VOID EFIAPI Udp6NetVectorExtFree (IN VOID * *Context*)

This function is a dummy ext-free function for the NET_BUF created for the output udp datagram.

Parameters:

[in] *Context* Pointer to the context data.

Referenced by Udp6Transmit().

VOID EFIAPI Udp6RecycleRxDataWrap (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

This function removes the Wrap specified by Context and releases relevant resources.

Parameters:

[in] *Event* The Event this notify function is registered to.
[in] *Context* Pointer to the context data.

This function removes the Wrap specified by Context and release relevant resources.

Parameters:

[in] *Event* The Event this notify function registered to.
[in] *Context* Pointer to the context data.

References _UDP6_RXDATA_WRAP::Link, _UDP6_RXDATA_WRAP::Packet, and _UDP6_RXDATA_WRAP::RxData.

Referenced by Udp6CheckTimeout(), Udp6FlushRcvdDgram(), and Udp6WrapRxData().

EFI_STATUS Udp6RemoveToken (IN NET_MAP * *TokenMap*,
IN EFI_UDP6_COMPLETION_TOKEN * *Token*
)

This function removes the specified Token from the TokenMap.

Parameters:

[in] *TokenMap* Pointer to the NET_MAP containing the tokens.
[in] *Token* Pointer to the Token to be removed.

Return values:

EFI_SUCCESS The specified Token is removed from the TokenMap.
EFI_NOT_FOUND The specified Token is not found in the TokenMap.

Referenced by Udp6DgramSent(), and Udp6Transmit().

VOID Udp6ReportIcmpError (IN UDP6_INSTANCE_DATA * *Instance*)

This function reports the received ICMP error.

Parameters:

[in] *Instance* Pointer to the udp instance context data.

Referenced by Udp6IcmpHandler(), and Udp6Receive().

```
VOID Udp6SendPortUnreach ( IN IP_IO *                    IpIo,
                           IN EFI_NET_SESSION_DATA *     NetSession,
                           IN VOID *                     Udp6Header
                         )
```

This function builds and sends out a icmp port unreachable message.

Parameters:

[in] *IpIo* Pointer to the IP_IO instance.

[in] *NetSession* Pointer to the EFI_NET_SESSION_DATA of the packet causes this icmp error message.

[in] *Udp6Header* Pointer to the udp header of the datagram causes this icmp error message.

Referenced by Udp6Demultiplex().

```
EFI_STATUS EFIAPI Udp6TokenExist ( IN NET_MAP *        Map,
                                   IN
                                   NET_MAP_ITEM        Item,
                                   *
                                   IN VOID *           Context
                                 )
```

This function checks whether the specified Token duplicates the one in the Map.

Parameters:

[in] *Map* Pointer to the NET_MAP.

[in] *Item* Pointer to the NET_MAP_ITEM contain the pointer to the Token.

[in] *Context* Pointer to the Token to be checked.

Return values:

EFI_SUCCESS The Token specified by Context differs from the one in the Item.

EFI_ACCESS_DENIED The Token duplicates with the one in the Item.

Referenced by Udp6Receive(), and Udp6Transmit().

```
EFI_STATUS Udp6ValidateTxToken ( IN UDP6_INSTANCE_DATA *          Instance,
                                 IN EFI_UDP6_COMPLETION_TOKEN *   TxToken
                               )
```

This function validates the TxToken. It returns the error code according to the spec.

Parameters:

[in] *Instance* Pointer to the udp instance context data.

[in] *TxToken* Pointer to the token to be checked.

Return values:

EFI_SUCCESS	The TxToken is valid.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: Token.Event is NULL; Token.Packet.TxData is NULL; Token.Packet.TxData.FragmentCount is zero; Token.Packet.TxData.DataLength is not equal to the sum of fragment lengths; One or more of the Token.Packet.TxData.FragmentTable[].FragmentLength fields is zero; One or more of the Token.Packet.TxData.FragmentTable[].FragmentBuffer fields is NULL; UdpSessionData.DestinationAddress are not valid unicast IPv6 addresses if the UdpSessionData is not NULL; UdpSessionData.DestinationPort and ConfigData.RemotePort are all zero if the UdpSessionData is not NULL.
EFI_BAD_BUFFER_SIZE	The data length is greater than the maximum UDP packet size.

References UDP6_MAX_DATA_SIZE.

Referenced by Udp6Transmit().

```
UDP6_RXDATA_WRAP * Udp6WrapRxData ( IN UDP6_INSTANCE_DATA *      Instance,
                                    IN NET_BUF *                 Packet,
                                    IN EFI_UDP6_RECEIVE_DATA *   RxData
                                  )
```

This function wraps the Packet into RxData.

Parameters:

[in]	*Instance*	Pointer to the instance context data.
[in]	*Packet*	Pointer to the buffer containing the received datagram.
[in]	*RxData*	Pointer to the EFI_UDP6_RECEIVE_DATA of this datagram.

Returns:

Pointer to the structure wrapping the RxData and the Packet.

References _UDP6_RXDATA_WRAP::Link, _UDP6_RXDATA_WRAP::Packet, _UDP6_RXDATA_WRAP::RxData, _UDP6_RXDATA_WRAP::TimeoutTick, and Udp6RecycleRxDataWrap().

Referenced by Udp6EnqueueDgram().

Variable Documentation

UINT16 mUdp6RandomPort

Referenced by Udp6Bind(), and Udp6DriverEntryPoint().

- File List
- Globals

NetworkPkg/Udp6Dxe/Udp6Impl.h File Reference

Data Structures

struct _UDP6_SERVICE_DA
struct _UDP6_INSTANCE_
struct _UDP6_RXDATA_W
struct UDP6_DESTROY_CH

Defines

"memItemRight" valign="bottom">UDP6_TIMEOUT_INTERVAL (50 * TICKS_PER_MS)
#define UDP6_HEADER_SIZ
#define UDP6_MAX_DATA_
#define UDP6_PORT_KNOW
#define UDP6_SERVICE_DA
('U', 'd', 'p', '6')
#define UDP6_INSTANCE_D
('U', 'd', 'p', 'S')
#define UDP6_SERVICE_DA
#define UDP6_INSTANCE_D

Typedefs

typedef struct _UDP6_SERVICE_DATA UDP6_SERVICE_DA
typedef struct _UDP6_INSTANCE_DATA UDP6_INSTANCE_D
typedef struct _UDP6_RXDATA_WRAP UDP6_RXDATA_WR

Functions

VOID Udp6CleanService (IN
*Udp6Service)
EFI_STATUS Udp6CreateService (IN
*Udp6Service, IN EFI
EFI_HANDLE Contro
VOID Udp6CleanInstance (IN
*Instance)
VOID Udp6InitInstance (IN U
*Udp6Service, IN OUT
*Instance)
VOID Udp6ReportIcmpError
*Instance)
EFI_STATUS EFIAPI Udp6GetModeData (IN
OUT EFI_UDP6_CON
EFI_IP6_MODE_DAT
EFI_MANAGED_NET
*MnpConfigData, OUT
*SnpModeData)
EFI_STATUS EFIAPI Udp6Configure (IN EF
EFI_UDP6_CONFIG_

EFI_STATUS EFIAPI Udp6Transmit (IN EFI_UDP6_
EFI_UDP6_COMPLETION_T
EFI_STATUS EFIAPI Udp6Receive (IN EFI_UDP6_
EFI_UDP6_COMPLETION_T
EFI_STATUS EFIAPI Udp6Cancel (IN EFI_UDP6_P
EFI_UDP6_COMPLETION_T
EFI_STATUS EFIAPI Udp6Poll (IN EFI_UDP6_PRO
EFI_STATUS EFIAPI Udp6Groups (IN EFI_UDP6_P
BOOLEAN JoinFlag, IN EFI_
*MulticastAddress)
EFI_STATUS Udp6Bind (IN LIST_ENTRY
EFI_UDP6_CONFIG_DATA
VOID Udp6BuildIp6ConfigData (IN
*Udp6ConfigData, IN OUT EF
*Ip6ConfigData)
EFI_STATUS EFIAPI Udp6TokenExist (IN NET_MA
NET_MAP_ITEM *Item, IN V
EFI_STATUS Udp6RemoveToken (IN NET_
EFI_UDP6_COMPLETION_T
BOOLEAN Udp6IsReconfigurable (IN EFI
*OldConfigData, IN EFI_UDP
*NewConfigData)
EFI_STATUS EFIAPI Udp6LeaveGroup (IN NET_M
NET_MAP_ITEM *Item, IN V
EFI_STATUS Udp6ValidateTxToken (IN UD
*Instance, IN EFI_UDP6_COM
*TxToken)
VOID EFIAPI Udp6NetVectorExtFree (IN VO
UINT16 Udp6Checksum (IN NET_BUF
HeadSum)
VOID Udp6InstanceDeliverDgram (I
*Instance)
EFI_STATUS Udp6InstanceCancelToken (IN
*Instance, IN EFI_UDP6_COM
VOID Udp6FlushRcvdDgram (IN UD
*Instance)

Variables

EFI_COMPONENT_NAME2_PROTOCOL gUdp6ComponentName2
EFI_COMPONENT_NAME_PROTOCOL gUdp6ComponentName
EFI_UNICODE_STRING_TABLE * gUdp6ControllerNameTable
EFI_SERVICE_BINDING_PROTOCOL mUdp6ServiceBinding
EFI_UDP6_PROTOCOL mUdp6Protocol
UINT16 mUdp6RandomPort

Detailed Description

Udp6 driver's whole implementation and internal data structures.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by Udp6Demultiplex(), and Udp6Transmit().

#define UDP6_INSTANCE_DATA_FROM_THIS (a)

Value:

```
CR ( \
  (a), \
  UDP6_INSTANCE_DATA, \
  Udp6Proto, \
  UDP6_INSTANCE_DATA_SIGNATURE \
  )
```

Referenced by Udp6Cancel(), Udp6Configure(), Udp6GetModeData(), Udp6Groups(), Udp6Poll(), Udp6Receive(), Udp6ServiceBindingDestroyChild(), and Udp6Transmit().

#define UDP6_INSTANCE_DATA_SIGNATURE SIGNATURE_32 ('U', 'd', 'p', 'S')

Referenced by Udp6CheckTimeout(), Udp6DestroyChildEntryInHandleBuffer(), and Udp6InitInstance().

#define UDP6_MAX_DATA_SIZE 65507

Referenced by Udp6ValidateTxToken().

#define UDP6_PORT_KNOWN 1024

Referenced by Udp6Bind(), and Udp6DriverEntryPoint().

#define UDP6_SERVICE_DATA_FROM_THIS (a)

Value:

```
CR ( \
  (a), \
  UDP6_SERVICE_DATA, \
  ServiceBinding, \
  UDP6_SERVICE_DATA_SIGNATURE \
  )
```

Referenced by Udp6DriverBindingStop(), Udp6ServiceBindingCreateChild(), and Udp6ServiceBindingDestroyChild().

#define UDP6_SERVICE_DATA_SIGNATURE SIGNATURE_32 ('U', 'd', 'p', '6')

Referenced by Udp6CheckTimeout(), and Udp6CreateService().

#define UDP6_TIMEOUT_INTERVAL (50 * TICKS_PER_MS)

Referenced by Udp6CheckTimeout(), and Udp6CreateService().

Typedef Documentation

typedef struct _UDP6_INSTANCE_DATA UDP6_INSTANCE_DATA

typedef struct _UDP6_RXDATA_WRAP UDP6_RXDATA_WRAP

typedef struct _UDP6_SERVICE_DATA UDP6_SERVICE_DATA

Function Documentation

```
EFI_STATUS Udp6Bind ( IN LIST_ENTRY *                 InstanceList,
                      IN EFI_UDP6_CONFIG_DATA *       ConfigData
                    )
```

This function tries to bind the udp instance according to the configured port allocation stragety.

Parameters:

[in]	*InstanceList*	Pointer to the head of the list linking the udp instances.
[in]	*ConfigData*	Pointer to the ConfigData of the instance to be bound.

Return values:

EFI_SUCCESS	The bound operation completed successfully.
EFI_ACCESS_DENIED	The <Address, Port> specified by the ConfigData is already used by another instance.
EFI_OUT_OF_RESOURCES	No available port resources.

This function tries to bind the udp instance according to the configured port allocation stragety.

Parameters:

[in]	*InstanceList*	Pointer to the head of the list linking the udp instances.
[in]	*ConfigData*	Pointer to the ConfigData of the instance to be bound.

Return values:

EFI_SUCCESS	The bound operation completed successfully.
EFI_ACCESS_DENIED	The <Address, Port> specified by the ConfigData is already used by other instance.
EFI_OUT_OF_RESOURCES	No available port resources.

References mUdp6RandomPort, UDP6_PORT_KNOWN, and Udp6FindInstanceByPort().

Referenced by Udp6Configure().

```
VOID Udp6BuildIp6ConfigData ( IN EFI_UDP6_CONFIG_DATA *      Udp6ConfigData,
                              IN OUT EFI_IP6_CONFIG_DATA *   Ip6ConfigData
                            )
```

This function builds the Ip6 configdata from the Udp6ConfigData.

Parameters:

[in]	*Udp6ConfigData*	Pointer to the EFI_UDP6_CONFIG_DATA.
[in,out]	*Ip6ConfigData*	Pointer to the EFI_IP6_CONFIG_DATA.

Referenced by Udp6Configure().

```
EFI_STATUS EFIAPI Udp6Cancel ( IN EFI_UDP6_PROTOCOL *                 This,
                               IN EFI_UDP6_COMPLETION_TOKEN *         Token
                             )
```

This function is used to abort a pending transmit or receive request.

Parameters:

[in]	*This*	Pointer to the EFI_UDP6_PROTOCOL instance.
[in]	*Token*	Pointer to a token that has been issued by EFI_UDP6_PROTOCOL.Transmit() or EFI_UDP6_PROTOCOL.Receive(). This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The asynchronous I/O request is aborted and Token.Event is signaled. When Token is NULL, all pending requests are aborted and their events are signaled.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This instance has not been started.
EFI_NO_MAPPING	When using the default address, configuration (DHCP, BOOTP, RARP, etc.) is not finished yet.
EFI_NOT_FOUND	When Token is not NULL, the asynchronous I/O request is not found in the transmit or receive queue. It either completed or was not issued by Transmit() or Receive().

This function is used to abort a pending transmit or receive request.

Parameters:

[in]	*This*	Pointer to the EFI_UDP6_PROTOCOL instance.
[in]	*Token*	Pointer to a token that has been issued by EFI_UDP6_PROTOCOL.Transmit() or EFI_UDP6_PROTOCOL.Receive(). This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The asynchronous I/O request was aborted, and Token.Event was signaled. When Token is NULL, all pending requests are aborted and their events are signaled.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This instance has not been started.
EFI_NO_MAPPING	When using the default address, configuration (DHCP, BOOTP, RARP, etc.) is not finished yet.

EFI_NOT_FOUND	When Token is not NULL, the asynchronous I/O request is not found in the transmit or receive queue. It is either completed or not issued by Transmit() or Receive().

References _UDP6_INSTANCE_DATA::Configured, UDP6_INSTANCE_DATA_FROM_THIS, and Udp6InstanceCancelToken().

UINT16 Udp6Checksum (IN NET_BUF * *Packet*,
IN UINT16 *HeadSum*
)

This function calculates the checksum for the Packet, utilizing the pre-calculated pseudo header to reduce overhead.

Parameters:

[in]	*Packet*	Pointer to the NET_BUF contains the udp datagram.
[in]	*HeadSum*	Checksum of the pseudo header execpt the length field.

Returns:

The 16-bit checksum of this udp datagram.

This function calculates the checksum for the Packet, utilizing the pre-calculated pseudo HeadSum to reduce some overhead.

Parameters:

[in]	*Packet*	Pointer to the NET_BUF contains the udp datagram.
[in]	*HeadSum*	Checksum of the pseudo header, execpt the length field.

Returns:

The 16-bit checksum of this udp datagram.

Referenced by Udp6Demultiplex(), and Udp6Transmit().

VOID Udp6CleanInstance (IN OUT UDP6_INSTANCE_DATA * *Instance*)

This function cleans the udp instance.

Parameters:

[in,out]	*Instance*	Pointer to the UDP6_INSTANCE_DATA to clean.

Referenced by Udp6ServiceBindingCreateChild(), and Udp6ServiceBindingDestroyChild().

VOID Udp6CleanService (IN OUT UDP6_SERVICE_DATA * *Udp6Service*)

Clean the Udp service context data.

Parameters:

[in,out]	*Udp6Service*	Pointer to the UDP6_SERVICE_DATA.

Referenced by Udp6DriverBindingStart(), and Udp6DriverBindingStop().

EFI_STATUS EFIAPI Udp6Configure (IN EFI_UDP6_PROTOCOL * *This*,

IN EFI_UDP6_CONFIG_DATA * *UdpConfigData*
)

This function is used to do the following: Initialize and start this instance of the EFI UDPv6 Protocol. Change the filtering rules and operational parameters. Reset this instance of the EFI UDPv6 Protocol.

Parameters:

[in]	*This*	Pointer to the EFI_UDP6_PROTOCOL instance.
[in]	*UdpConfigData*	Pointer to the buffer to set the configuration data. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The configuration settings were set, changed, or reset successfully.
EFI_NO_MAPPING	When the UdpConifgData.UseAnyStationAddress is set to true and there is no address available for IP6 driver to binding source address to this instance.
EFI_INVALID_PARAMETER	One or more following conditions are TRUE: This is NULL. UdpConfigData.StationAddress is not a valid unicast IPv6 address. UdpConfigData.RemoteAddress is not a valid unicast IPv6 address, if it is not zero.
EFI_ALREADY_STARTED	The EFI UDPv6 Protocol instance is already started/configured and must be stopped/reset before it can be reconfigured. Only TrafficClass, HopLimit, ReceiveTimeout, and TransmitTimeout can be reconfigured without stopping the current instance of the EFI UDPv6 Protocol.
EFI_ACCESS_DENIED	UdpConfigData.AllowDuplicatePort is FALSE, and UdpConfigData.StationPort is already used by another instance.
EFI_OUT_OF_RESOURCES	The EFI UDPv6 Protocol driver cannot allocate memory for this EFI UDPv6 Protocol instance.
EFI_DEVICE_ERROR	An unexpected network or system error occurred, and this instance was not opened.

This function is used to do the following: Initialize and start this instance of the EFI UDPv6 Protocol. Change the filtering rules and operational parameters. Reset this instance of the EFI UDPv6 Protocol.

Parameters:

[in]	*This*	Pointer to the EFI_UDP6_PROTOCOL instance.
[in]	*UdpConfigData*	Pointer to the buffer to set the configuration data. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The configuration settings were set, changed, or reset successfully.
EFI_NO_MAPPING	When the UdpConifgData.UseAnyStationAddress is set to true and there is no address available for the IP6 driver to bind a source address to this instance.
EFI_INVALID_PARAMETER	One or more following conditions are TRUE: This is NULL. UdpConfigData.StationAddress is not a valid unicast IPv6 address. UdpConfigData.RemoteAddress is not a valid unicast IPv6 address if it is not zero.
EFI_ALREADY_STARTED	The EFI UDPv6 Protocol instance is already started/configured and must be stopped/reset before it can be reconfigured. Only TrafficClass, HopLimit, ReceiveTimeout, and TransmitTimeout can be reconfigured without stopping the current instance of the EFI UDPv6 Protocol.

EFI_ACCESS_DENIED	UdpConfigData.AllowDuplicatePort is FALSE and UdpConfigData.StationPort is already used by another instance.
EFI_OUT_OF_RESOURCES	The EFI UDPv6 Protocol driver cannot allocate memory for this EFI UDPv6 Protocol instance.
EFI_DEVICE_ERROR	An unexpected network or system error occurred, and this instance was not opened.

References _UDP6_SERVICE_DATA::ChildrenList, _UDP6_INSTANCE_DATA::ConfigData, _UDP6_INSTANCE_DATA::Configured, _UDP6_INSTANCE_DATA::DeliveredDgramQue, _UDP6_INSTANCE_DATA::HeadSum, _UDP6_INSTANCE_DATA::IpInfo, _UDP6_INSTANCE_DATA::IsNoMapping, UDP6_INSTANCE_DATA_FROM_THIS, Udp6Bind(), Udp6BuildIp6ConfigData(), Udp6FlushRcvdDgram(), Udp6IsReconfigurable(), _UDP6_INSTANCE_DATA::Udp6Proto, and _UDP6_INSTANCE_DATA::Udp6Service.

```
EFI_STATUS Udp6CreateService ( IN UDP6_SERVICE_DATA *   Udp6Service,
                               IN EFI_HANDLE            ImageHandle,
                               IN EFI_HANDLE            ControllerHandle
                             )
```

Create the Udp service context data.

Parameters:

[in]	*Udp6Service*	Pointer to the UDP6_SERVICE_DATA.
[in]	*ImageHandle*	The image handle of this udp6 driver.
[in]	*ControllerHandle*	The controller handle this udp6 driver binds on.

Return values:

EFI_SUCCESS	The udp6 service context data was created and initialized.
EFI_OUT_OF_RESOURCES	Cannot allocate memory.
Others	An error condition occurred.

Create the Udp service context data.

Parameters:

[in]	*Udp6Service*	Pointer to the UDP6_SERVICE_DATA.
[in]	*ImageHandle*	The image handle of this udp6 driver.
[in]	*ControllerHandle*	The controller handle this udp6 driver binds on.

Return values:

EFI_SUCCESS	The udp6 service context data was created and initialized.
EFI_OUT_OF_RESOURCES	Cannot allocate memory.
Others	An error condition occurred.

References mUdp6ServiceBinding, UDP6_SERVICE_DATA_SIGNATURE, UDP6_TIMEOUT_INTERVAL, Udp6CheckTimeout(), Udp6DgramRcvd(), and Udp6DgramSent().

Referenced by Udp6DriverBindingStart().

```
VOID Udp6FlushRcvdDgram ( IN UDP6_INSTANCE_DATA *   Instance   )
```

This function removes all the Wrap datas in the RcvdDgramQue.

Parameters:

[in] *Instance* Pointer to the Udp6 Instance.

References Udp6RecycleRxDataWrap().

Referenced by Udp6Configure().

```
EFI_STATUS EFIAPI
Udp6GetModeData
( IN EFI_UDP6_PROTOCOL *                    This,
  OUT EFI_UDP6_CONFIG_DATA *                Udp6ConfigData,
  OUT EFI_IP6_MODE_DATA *                   Ip6ModeData,
  OUT
  EFI_MANAGED_NETWORK_CONFIG_DATA *         MnpConfigData,
  OUT EFI_SIMPLE_NETWORK_MODE *             SnpModeData
)
```

This function copies the current operational settings of this EFI UDPv6 Protocol instance into user-supplied buffers. This function is used optionally to retrieve the operational mode data of underlying networks or drivers.

Parameters:

[in]	*This*	Pointer to the EFI_UDP6_PROTOCOL instance.
[out]	*Udp6ConfigData*	The buffer in which the current UDP configuration data is returned. This parameter is optional and may be NULL.
[out]	*Ip6ModeData*	The buffer in which the current EFI IPv6 Protocol mode data is returned. This parameter is optional and may be NULL.
[out]	*MnpConfigData*	The buffer in which the current managed network configuration data is returned. This parameter is optional and may be NULL.
[out]	*SnpModeData*	The buffer in which the simple network mode data is returned. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The mode data was read.
EFI_NOT_STARTED	When Udp6ConfigData is queried, no configuration data is available because this instance has not been started.
EFI_INVALID_PARAMETER	This is NULL.

This function copies the current operational settings of this EFI UDPv6 Protocol instance into user-supplied buffers. This function is used optionally to retrieve the operational mode data of underlying networks or drivers.

Parameters:

[in]	*This*	Pointer to the EFI_UDP6_PROTOCOL instance.
[out]	*Udp6ConfigData*	The buffer in which the current UDP configuration data is returned. This parameter is optional and may be NULL.
[out]	*Ip6ModeData*	The buffer in which the current EFI IPv6 Protocol mode data is returned. This parameter is optional and may be NULL.
[out]	*MnpConfigData*	The buffer in which the current managed network configuration data is returned. This parameter is optional and may be NULL.
[out]	*SnpModeData*	The buffer in which the simple network mode data is returned. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The mode data was read.
EFI_NOT_STARTED	When Udp6ConfigData is queried, no configuration data is available because this instance has not been started.
EFI_INVALID_PARAMETER	This is NULL.

References _UDP6_INSTANCE_DATA::ConfigData, _UDP6_INSTANCE_DATA::Configured, _UDP6_INSTANCE_DATA::IpInfo, and UDP6_INSTANCE_DATA_FROM_THIS.

```
EFI_STATUS EFIAPI Udp6Groups ( IN EFI_UDP6_PROTOCOL *   This,
                               IN BOOLEAN               JoinFlag,
                               IN EFI_IPv6_ADDRESS *    MulticastAddress
                             )
```

This function is used to enable and disable the multicast group filtering.

Parameters:

[in] *This*	Pointer to the EFI_UDP6_PROTOCOL instance.
[in] *JoinFlag*	Set to TRUE to join a multicast group. Set to FALSE to leave one or all multicast groups.
[in] *MulticastAddress*	Pointer to multicast group address to join or leave. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_STARTED	The EFI UDPv6 Protocol instance has not been started.
EFI_OUT_OF_RESOURCES	Could not allocate resources to join the group.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. JoinFlag is TRUE and MulticastAddress is NULL. JoinFlag is TRUE and MulticastAddress is not a valid multicast address.
EFI_ALREADY_STARTED	The group address is already in the group table (when JoinFlag is TRUE).
EFI_NOT_FOUND	The group address is not in the group table (when JoinFlag is FALSE).
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

This function is used to enable and disable the multicast group filtering.

Parameters:

[in] *This*	Pointer to the EFI_UDP6_PROTOCOL instance.
[in] *JoinFlag*	Set to TRUE to join a multicast group. Set to FALSE to leave one or all multicast groups.
[in] *MulticastAddress*	Pointer to multicast group address to join or leave. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_STARTED	The EFI UDPv6 Protocol instance has not been started.
EFI_OUT_OF_RESOURCES	Could not allocate resources to join the group.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. JoinFlag is TRUE and MulticastAddress is NULL. JoinFlag is TRUE and *MulticastAddress is not a valid multicast address.

EFI_ALREADY_STARTED	The group address is already in the group table (when JoinFlag is TRUE).
EFI_NOT_FOUND	The group address is not in the group table (when JoinFlag is FALSE).
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References _UDP6_INSTANCE_DATA::Configured, _UDP6_INSTANCE_DATA::IpInfo, _UDP6_INSTANCE_DATA::McastIps, UDP6_INSTANCE_DATA_FROM_THIS, and Udp6LeaveGroup().

```
VOID Udp6InitInstance ( IN UDP6_SERVICE_DATA *          Udp6Service,
                        IN OUT UDP6_INSTANCE_DATA *     Instance
                      )
```

This function intializes the new created udp instance.

Parameters:

`[in]`	*Udp6Service*	Pointer to the UDP6_SERVICE_DATA.
`[in,out]`	*Instance*	Pointer to the un-initialized UDP6_INSTANCE_DATA.

This function intializes the new created udp instance.

Parameters:

`[in]`	*Udp6Service*	Pointer to the UDP6_SERVICE_DATA.
`[in,out]`	*Instance*	Pointer to the un-initialized UDP6_INSTANCE_DATA.

References mUdp6Protocol, and UDP6_INSTANCE_DATA_SIGNATURE.

Referenced by Udp6ServiceBindingCreateChild().

```
EFI_STATUS Udp6InstanceCancelToken ( IN UDP6_INSTANCE_DATA *            Instance,
                                     IN EFI_UDP6_COMPLETION_TOKEN *     Token
                                   )
```

Cancel Udp6 tokens from the Udp6 instance.

Parameters:

`[in]`	*Instance*	Pointer to the udp instance context data.
`[in]`	*Token*	Pointer to the token to be canceled. If NULL, all tokens in this instance will be cancelled. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The Token is cancelled.
EFI_NOT_FOUND	The Token is not found.

References Udp6CancelTokens().

Referenced by Udp6Cancel().

```
VOID Udp6InstanceDeliverDgram ( IN UDP6_INSTANCE_DATA *   Instance   )
```

This function delivers the received datagrams to the specified instance.

Parameters:

[in] *Instance* Pointer to the instance context data.

References _UDP6_RXDATA_WRAP::Link, _UDP6_RXDATA_WRAP::Packet, and _UDP6_RXDATA_WRAP::RxData.

Referenced by Udp6DeliverDgram(), and Udp6Receive().

```
BOOLEAN Udp6IsReconfigurable ( IN EFI_UDP6_CONFIG_DATA *  OldConfigData,
                               IN EFI_UDP6_CONFIG_DATA *  NewConfigData
                             )
```

This function is used to check whether the NewConfigData has any un-reconfigurable parameters changed compared to the OldConfigData.

Parameters:

[in] *OldConfigData* Pointer to the current ConfigData the udp instance uses.

[in] *NewConfigData* Pointer to the new ConfigData.

Return values:

TRUE The instance is reconfigurable according to NewConfigData.

FALSE The instance is not reconfigurable according to NewConfigData.

This function is used to check whether the NewConfigData has any un-reconfigurable parameters changed compared to the OldConfigData.

Parameters:

[in] *OldConfigData* Pointer to the current ConfigData the udp instance uses.

[in] *NewConfigData* Pointer to the new ConfigData.

Return values:

TRUE The instance is reconfigurable according to the NewConfigData.

FALSE Otherwise.

Referenced by Udp6Configure().

```
EFI_STATUS EFIAPI Udp6LeaveGroup ( IN NET_MAP *          Map,
                                   IN NET_MAP_ITEM *     Item,
                                   IN VOID *             Arg
                                 )
```

This function removes the multicast group specified by Arg from the Map.

Parameters:

[in] *Map* Pointer to the NET_MAP.

[in] *Item* Pointer to the NET_MAP_ITEM.

[in] *Arg* Pointer to the Arg. It is the pointer to a multicast IPv6 Address. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS The multicast address is removed.

EFI_ABORTED The specified multicast address is removed, and the Arg is not NULL.

This function removes the multicast group specified by Arg from the Map.

Parameters:

[in] *Map* Pointer to the NET_MAP.
[in] *Item* Pointer to the NET_MAP_ITEM.
[in] *Arg* Pointer to the Arg, it's the pointer to a multicast IPv6 Address. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS The multicast address is removed.
EFI_ABORTED The specified multicast address is removed, and the Arg is not NULL.

Referenced by Udp6Groups().

VOID EFIAPI Udp6NetVectorExtFree (IN VOID * *Context*)

This function is a dummy ext-free function for the NET_BUF created for the output udp datagram.

Parameters:

[in] *Context* Pointer to the context data.

This function is a dummy ext-free function for the NET_BUF created for the output udp datagram.

Parameters:

[in] *Context* Pointer to the context data.

Referenced by Udp6Transmit().

EFI_STATUS EFIAPI Udp6Poll (IN EFI_UDP6_PROTOCOL * *This*)

This function can be used by network drivers and applications to increase the rate that data packets are moved between the communications device and the transmit/receive queues.

Parameters:

[in] *This* Pointer to the EFI_UDP6_PROTOCOL instance.

Return values:

EFI_SUCCESS Incoming or outgoing data was processed.
EFI_INVALID_PARAMETER This is NULL.
EFI_DEVICE_ERROR An unexpected system or network error occurred.
EFI_TIMEOUT Data was dropped out of the transmit and/or receive queue.

References _UDP6_INSTANCE_DATA::IpInfo, and UDP6_INSTANCE_DATA_FROM_THIS.

EFI_STATUS EFIAPI Udp6Receive (IN EFI_UDP6_PROTOCOL * *This*,
IN EFI_UDP6_COMPLETION_TOKEN * *Token*
)

This function places a completion token into the receive packet queue. This function is always asynchronous.

Parameters:

[in] *This* Pointer to the EFI_UDP6_PROTOCOL instance.
[in] *Token* Pointer to a token that is associated with the receive data descriptor.

Return values:

EFI_SUCCESS	The receive completion token is cached.
EFI_NOT_STARTED	This EFI UDPv6 Protocol instance has not been started.
EFI_NO_MAPPING	When using a default address, configuration (DHCP, BOOTP, RARP, etc.) is not finished yet.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. Token is NULL. Token.Event is NULL.
EFI_OUT_OF_RESOURCES	The receive completion token could not be queued due to a lack of system resources (usually memory).
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI UDPv6 Protocol instance has been reset to startup defaults.
EFI_ACCESS_DENIED	A receive completion token with the same Token.Event is already in the receive queue.
EFI_NOT_READY	The receive request could not be queued because the receive queue is full.

This function places a completion token into the receive packet queue. This function is always asynchronous.

Parameters:

[in]	*This*	Pointer to the EFI_UDP6_PROTOCOL instance.
[in]	*Token*	Pointer to a token that is associated with the receive data descriptor.

Return values:

EFI_SUCCESS	The receive completion token was cached.
EFI_NOT_STARTED	This EFI UDPv6 Protocol instance has not been started.
EFI_NO_MAPPING	When using a default address, configuration (DHCP, BOOTP, RARP, etc.) is not finished yet.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. Token is NULL. Token.Event is NULL.
EFI_OUT_OF_RESOURCES	The receive completion token could not be queued due to a lack of system resources (usually memory).
EFI_DEVICE_ERROR	An unexpected system or network error occurred. The EFI UDPv6 Protocol instance has been reset to startup defaults.
EFI_ACCESS_DENIED	A receive completion token with the same Token.Event is already in the receive queue.
EFI_NOT_READY	The receive request could not be queued because the receive queue is full.

References _UDP6_INSTANCE_DATA::Configured, _UDP6_INSTANCE_DATA::RxTokens, _UDP6_INSTANCE_DATA::TxTokens, UDP6_INSTANCE_DATA_FROM_THIS, Udp6InstanceDeliverDgram(), Udp6ReportIcmpError(), and Udp6TokenExist().

```
EFI_STATUS Udp6RemoveToken ( IN NET_MAP *                    TokenMap,
                             IN EFI_UDP6_COMPLETION_TOKEN *  Token
                           )
```

This function removes the specified Token from the TokenMap.

Parameters:

[in]	*TokenMap*	Pointer to the NET_MAP containing the tokens.
[in]	*Token*	Pointer to the Token to be removed.

Return values:

EFI_SUCCESS The specified Token is removed from the TokenMap.
EFI_NOT_FOUND The specified Token is not found in the TokenMap.

Referenced by Udp6DgramSent(), and Udp6Transmit().

VOID Udp6ReportIcmpError (IN UDP6_INSTANCE_DATA * *Instance*)

This function reports the received ICMP error.

Parameters:

[in] *Instance* Pointer to the udp instance context data.

Referenced by Udp6IcmpHandler(), and Udp6Receive().

```
EFI_STATUS EFIAPI Udp6TokenExist ( IN NET_MAP *          Map,
                                   IN NET_MAP_ITEM *     Item,
                                   IN VOID *             Context
                                 )
```

This function checks whether the specified Token duplicates with the one in the Map.

Parameters:

[in] *Map* Pointer to the NET_MAP.
[in] *Item* Pointer to the NET_MAP_ITEM contain the pointer to the Token.
[in] *Context* Pointer to the Token to be checked.

Return values:

EFI_SUCCESS The Token specified by Context differs from the one in the Item.
EFI_ACCESS_DENIED The Token duplicates with the one in the Item.

This function checks whether the specified Token duplicates the one in the Map.

Parameters:

[in] *Map* Pointer to the NET_MAP.
[in] *Item* Pointer to the NET_MAP_ITEM contain the pointer to the Token.
[in] *Context* Pointer to the Token to be checked.

Return values:

EFI_SUCCESS The Token specified by Context differs from the one in the Item.
EFI_ACCESS_DENIED The Token duplicates with the one in the Item.

Referenced by Udp6Receive(), and Udp6Transmit().

```
EFI_STATUS EFIAPI Udp6Transmit ( IN EFI_UDP6_PROTOCOL *                 This,
                                 IN EFI_UDP6_COMPLETION_TOKEN *         Token
                               )
```

This function places a sending request to this instance of the EFI UDPv6 Protocol, alongside the transmit data that was filled by the user.

Parameters:

[in] *This* Pointer to the EFI_UDP6_PROTOCOL instance.

[in] *Token* Pointer to the completion token that will be placed into the transmit queue.

Return values:

EFI_SUCCESS	The data has been queued for transmission.
EFI_NOT_STARTED	This EFI UDPv6 Protocol instance has not been started.
EFI_NO_MAPPING	The under-layer IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: This is NULL. Token is NULL. Token.Event is NULL. Token.Packet.TxData is NULL. Token.Packet.TxData.FragmentCount is zero. Token.Packet.TxData.DataLength is not equal to the sum of fragment lengths. One or more of the Token.Packet.TxData.FragmentTable[] .FragmentLength fields is zero. One or more of the Token.Packet.TxData.FragmentTable[] .FragmentBuffer fields is NULL. One or more of the Token.Packet.TxData.UdpSessionData. DestinationAddres are not valid unicast IPv6 addresses, if the UdpSessionData is not NULL. Token.Packet.TxData.UdpSessionData. DestinationAddres is NULL Token.Packet.TxData.UdpSessionData. DestinatioPort is zero. Token.Packet.TxData.UdpSessionData is NULL and this instance's UdpConfigData.RemoteAddress is unspecified.
EFI_ACCESS_DENIED	The transmit completion token with the same Token.Event is already in the transmit queue.
EFI_NOT_READY	The completion token could not be queued because the transmit queue is full.
EFI_OUT_OF_RESOURCES	Could not queue the transmit data.
EFI_NOT_FOUND	There is no route to the destination network or address.
EFI_BAD_BUFFER_SIZE	The data length is greater than the maximum UDP packet size. Or the length of the IP header + UDP header + data length is greater than MTU if DoNotFragment is TRUE.

This function places a sending request to this instance of the EFI UDPv6 Protocol, alongside the transmit data that was filled by the user.

Parameters:

[in] *This* Pointer to the EFI_UDP6_PROTOCOL instance.

[in] *Token* Pointer to the completion token that will be placed into the transmit queue.

Return values:

EFI_SUCCESS	The data was queued for transmission.
EFI_NOT_STARTED	This EFI UDPv6 Protocol instance has not been started.
EFI_NO_MAPPING	The under-layer IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: This is NULL. Token is NULL. Token.Event is NULL. Token.Packet.TxData is NULL. Token.Packet.TxData.FragmentCount is zero. Token.Packet.TxData.DataLength is not equal to the sum of fragment lengths. One or more of the Token.Packet.TxData.FragmentTable[].FragmentLength fields is zero. One or more of the Token.Packet.TxData.FragmentTable[].FragmentBuffer fields is

	NULL. One or more of the Token.Packet.TxData.UdpSessionData.DestinationAddres are not valid unicast IPv6 addresses if the UdpSessionData is not NULL. Token.Packet.TxData.UdpSessionData. DestinationAddress is NULL Token.Packet.TxData.UdpSessionData. DestinatioPort is zero. Token.Packet.TxData.UdpSessionData is NULL and this instance's UdpConfigData.RemoteAddress is unspecified.
EFI_ACCESS_DENIED	The transmit completion token with the same Token.Event is already in the transmit queue.
EFI_NOT_READY	The completion token could not be queued because the transmit queue is full.
EFI_OUT_OF_RESOURCES	Could not queue the transmit data.
EFI_NOT_FOUND	There is no route to the destination network or address.
EFI_BAD_BUFFER_SIZE	The data length is greater than the maximum UDP packet size. Or, the length of the IP header + UDP header + data length is greater than MTU if DoNotFragment is TRUE.

References _UDP6_INSTANCE_DATA::ConfigData, _UDP6_INSTANCE_DATA::Configured, _UDP6_INSTANCE_DATA::HeadSum, _UDP6_INSTANCE_DATA::IpInfo, _UDP6_SERVICE_DATA::IpIo, _UDP6_INSTANCE_DATA::RxTokens, _UDP6_INSTANCE_DATA::TxTokens, UDP6_HEADER_SIZE, UDP6_INSTANCE_DATA_FROM_THIS, Udp6Checksum(), Udp6NetVectorExtFree(), Udp6RemoveToken(), _UDP6_INSTANCE_DATA::Udp6Service, Udp6TokenExist(), and Udp6ValidateTxToken().

```
EFI_STATUS Udp6ValidateTxToken ( IN UDP6_INSTANCE_DATA *          Instance,
                                 IN EFI_UDP6_COMPLETION_TOKEN *   TxToken
                               )
```

This function validates the TxToken, it returns the error code according to the spec.

Parameters:

[in] *Instance*	Pointer to the udp instance context data.
[in] *TxToken*	Pointer to the token to be checked.

Return values:

EFI_SUCCESS	The TxToken is valid.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: Token.Event is NULL. Token.Packet.TxData is NULL. Token.Packet.TxData.FragmentCount is zero. Token.Packet.TxData.DataLength is not equal to the sum of fragment lengths. One or more of the Token.Packet.TxData.FragmentTable[].FragmentLength fields is zero. One or more of the Token.Packet.TxData.FragmentTable[].FragmentBuffer fields is NULL. UdpSessionData.DestinationAddress are not valid unicast IPv6 addresses if the UdpSessionData is not NULL. UdpSessionData.DestinationPort and ConfigData.RemotePort are all zero if the UdpSessionData is not NULL.
EFI_BAD_BUFFER_SIZE	The data length is greater than the maximum UDP packet size.

This function validates the TxToken. It returns the error code according to the spec.

Parameters:

[in]	*Instance*	Pointer to the udp instance context data.
[in]	*TxToken*	Pointer to the token to be checked.

Return values:

EFI_SUCCESS	The TxToken is valid.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: Token.Event is NULL; Token.Packet.TxData is NULL; Token.Packet.TxData.FragmentCount is zero; Token.Packet.TxData.DataLength is not equal to the sum of fragment lengths; One or more of the Token.Packet.TxData.FragmentTable[].FragmentLength fields is zero; One or more of the Token.Packet.TxData.FragmentTable[].FragmentBuffer fields is NULL; UdpSessionData.DestinationAddress are not valid unicast IPv6 addresses if the UdpSessionData is not NULL; UdpSessionData.DestinationPort and ConfigData.RemotePort are all zero if the UdpSessionData is not NULL.
EFI_BAD_BUFFER_SIZE	The data length is greater than the maximum UDP packet size.

References UDP6_MAX_DATA_SIZE.

Referenced by Udp6Transmit().

Variable Documentation

EFI_COMPONENT_NAME_PROTOCOL gUdp6ComponentName

Referenced by Udp6ComponentNameGetControllerName(), Udp6ComponentNameGetDriverName(), Udp6DriverEntryPoint(), and UpdateName().

EFI_COMPONENT_NAME2_PROTOCOL gUdp6ComponentName2

Referenced by Udp6DriverEntryPoint(), and UpdateName().

EFI_UNICODE_STRING_TABLE* gUdp6ControllerNameTable

Referenced by Udp6ComponentNameGetControllerName(), and UpdateName().

EFI_UDP6_PROTOCOL mUdp6Protocol

Referenced by Udp6InitInstance().

UINT16 mUdp6RandomPort

Referenced by Udp6Bind(), and Udp6DriverEntryPoint().

EFI_SERVICE_BINDING_PROTOCOL mUdp6ServiceBinding

Referenced by Udp6CreateService().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/Udp6Dxe/Udp6Main.c File Reference

Functions

EFI_STATUS EFIAPI Udp6GetModeData (IN EFI_UDP6_PROTOCOL *This, OUT EFI_UDP6_CONFIG_DATA *Udp6ConfigData, OUT EFI_IP6_MODE_DATA *Ip6ModeData, OUT EFI_MANAGED_NETWORK_CONFIG_DATA *MnpConfigData, OUT EFI_SIMPLE_NETWORK_MODE *SnpModeData)
EFI_STATUS EFIAPI Udp6Configure (IN EFI_UDP6_PROTOCOL *This, IN EFI_UDP6_CONFIG_DATA *UdpConfigData)
EFI_STATUS EFIAPI Udp6Groups (IN EFI_UDP6_PROTOCOL *This, IN BOOLEAN JoinFlag, IN EFI_IPv6_ADDRESS *MulticastAddress)
EFI_STATUS EFIAPI Udp6Transmit (IN EFI_UDP6_PROTOCOL *This, IN EFI_UDP6_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Udp6Receive (IN EFI_UDP6_PROTOCOL *This, IN EFI_UDP6_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Udp6Cancel (IN EFI_UDP6_PROTOCOL *This, IN EFI_UDP6_COMPLETION_TOKEN *Token)
EFI_STATUS EFIAPI Udp6Poll (IN EFI_UDP6_PROTOCOL *This)

Variables

EFI_UDP6_PROTOCOL mUdp6Protocol

Detailed Description

Contains all EFI_UDP6_PROTOCOL interfaces.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI Udp6Cancel ( IN EFI_UDP6_PROTOCOL *          This,
                               IN EFI_UDP6_COMPLETION_TOKEN *  Token
                             )
```

This function is used to abort a pending transmit or receive request.

Parameters:

- [in] *This* Pointer to the EFI_UDP6_PROTOCOL instance.
- [in] *Token* Pointer to a token that has been issued by EFI_UDP6_PROTOCOL.Transmit() or

EFI_UDP6_PROTOCOL.Receive(). This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The asynchronous I/O request was aborted, and Token.Event was signaled. When Token is NULL, all pending requests are aborted and their events are signaled.
EFI_INVALID_PARAMETER	This is NULL.
EFI_NOT_STARTED	This instance has not been started.
EFI_NO_MAPPING	When using the default address, configuration (DHCP, BOOTP, RARP, etc.) is not finished yet.
EFI_NOT_FOUND	When Token is not NULL, the asynchronous I/O request is not found in the transmit or receive queue. It is either completed or not issued by Transmit() or Receive().

References _UDP6_INSTANCE_DATA::Configured, UDP6_INSTANCE_DATA_FROM_THIS, and Udp6InstanceCancelToken().

```
EFI_STATUS EFIAPI Udp6Configure ( IN EFI_UDP6_PROTOCOL *        This,
                                  IN EFI_UDP6_CONFIG_DATA *     UdpConfigData
                                )
```

This function is used to do the following: Initialize and start this instance of the EFI UDPv6 Protocol. Change the filtering rules and operational parameters. Reset this instance of the EFI UDPv6 Protocol.

Parameters:

`[in]` *This*	Pointer to the EFI_UDP6_PROTOCOL instance.
`[in]` *UdpConfigData*	Pointer to the buffer to set the configuration data. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The configuration settings were set, changed, or reset successfully.
EFI_NO_MAPPING	When the UdpConifgData.UseAnyStationAddress is set to true and there is no address available for the IP6 driver to bind a source address to this instance.
EFI_INVALID_PARAMETER	One or more following conditions are TRUE: This is NULL. UdpConfigData.StationAddress is not a valid unicast IPv6 address. UdpConfigData.RemoteAddress is not a valid unicast IPv6 address if it is not zero.
EFI_ALREADY_STARTED	The EFI UDPv6 Protocol instance is already started/configured and must be stopped/reset before it can be reconfigured. Only TrafficClass, HopLimit, ReceiveTimeout, and TransmitTimeout can be reconfigured without stopping the current instance of the EFI UDPv6 Protocol.
EFI_ACCESS_DENIED	UdpConfigData.AllowDuplicatePort is FALSE and UdpConfigData.StationPort is already used by another instance.
EFI_OUT_OF_RESOURCES	The EFI UDPv6 Protocol driver cannot allocate memory for this EFI UDPv6 Protocol instance.
EFI_DEVICE_ERROR	An unexpected network or system error occurred, and this instance was not opened.

References _UDP6_SERVICE_DATA::ChildrenList, _UDP6_INSTANCE_DATA::ConfigData, _UDP6_INSTANCE_DATA::Configured, _UDP6_INSTANCE_DATA::DeliveredDgramQue, _UDP6_INSTANCE_DATA::HeadSum, _UDP6_INSTANCE_DATA::IpInfo,

_UDP6_INSTANCE_DATA::IsNoMapping, UDP6_INSTANCE_DATA_FROM_THIS, Udp6Bind(), Udp6BuildIp6ConfigData(), Udp6FlushRcvdDgram(), Udp6IsReconfigurable(), _UDP6_INSTANCE_DATA::Udp6Proto, and _UDP6_INSTANCE_DATA::Udp6Service.

```
EFI_STATUS EFIAPI
Udp6GetModeData ( IN EFI_UDP6_PROTOCOL *                  This,
                  OUT EFI_UDP6_CONFIG_DATA *              Udp6ConfigData,
                  OUT EFI_IP6_MODE_DATA *                 Ip6ModeData,
                  OUT
                  EFI_MANAGED_NETWORK_CONFIG_DATA *       MnpConfigData,
                  OUT EFI_SIMPLE_NETWORK_MODE *           SnpModeData
                  )
```

This function copies the current operational settings of this EFI UDPv6 Protocol instance into user-supplied buffers. This function is used optionally to retrieve the operational mode data of underlying networks or drivers.

Parameters:

[in]	*This*	Pointer to the EFI_UDP6_PROTOCOL instance.
[out]	*Udp6ConfigData*	The buffer in which the current UDP configuration data is returned. This parameter is optional and may be NULL.
[out]	*Ip6ModeData*	The buffer in which the current EFI IPv6 Protocol mode data is returned. This parameter is optional and may be NULL.
[out]	*MnpConfigData*	The buffer in which the current managed network configuration data is returned. This parameter is optional and may be NULL.
[out]	*SnpModeData*	The buffer in which the simple network mode data is returned. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The mode data was read.
EFI_NOT_STARTED	When Udp6ConfigData is queried, no configuration data is available because this instance has not been started.
EFI_INVALID_PARAMETER	This is NULL.

References _UDP6_INSTANCE_DATA::ConfigData, _UDP6_INSTANCE_DATA::Configured, _UDP6_INSTANCE_DATA::IpInfo, and UDP6_INSTANCE_DATA_FROM_THIS.

```
EFI_STATUS EFIAPI Udp6Groups ( IN EFI_UDP6_PROTOCOL *   This,
                               IN BOOLEAN               JoinFlag,
                               IN EFI_IPv6_ADDRESS *    MulticastAddress
                               )
```

This function is used to enable and disable the multicast group filtering.

Parameters:

[in]	*This*	Pointer to the EFI_UDP6_PROTOCOL instance.
[in]	*JoinFlag*	Set to TRUE to join a multicast group. Set to FALSE to leave one or all multicast groups.
[in]	*MulticastAddress*	Pointer to multicast group address to join or leave. This parameter is optional and may be NULL.

Return values:

EFI_SUCCESS	The operation completed successfully.
EFI_NOT_STARTED	The EFI UDPv6 Protocol instance has not been started.
EFI_OUT_OF_RESOURCES	Could not allocate resources to join the group.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. JoinFlag is TRUE and MulticastAddress is NULL. JoinFlag is TRUE and *MulticastAddress is not a valid multicast address.
EFI_ALREADY_STARTED	The group address is already in the group table (when JoinFlag is TRUE).
EFI_NOT_FOUND	The group address is not in the group table (when JoinFlag is FALSE).
EFI_DEVICE_ERROR	An unexpected system or network error occurred.

References _UDP6_INSTANCE_DATA::Configured, _UDP6_INSTANCE_DATA::IpInfo, _UDP6_INSTANCE_DATA::McastIps, UDP6_INSTANCE_DATA_FROM_THIS, and Udp6LeaveGroup().

EFI_STATUS EFIAPI Udp6Poll (IN EFI_UDP6_PROTOCOL * *This*)

This function can be used by network drivers and applications to increase the rate that data packets are moved between the communications device and the transmit/receive queues.

Parameters:

`[in]` *This* Pointer to the EFI_UDP6_PROTOCOL instance.

Return values:

EFI_SUCCESS	Incoming or outgoing data was processed.
EFI_INVALID_PARAMETER	This is NULL.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
EFI_TIMEOUT	Data was dropped out of the transmit and/or receive queue.

References _UDP6_INSTANCE_DATA::IpInfo, and UDP6_INSTANCE_DATA_FROM_THIS.

EFI_STATUS EFIAPI Udp6Receive (IN EFI_UDP6_PROTOCOL * *This*,
IN EFI_UDP6_COMPLETION_TOKEN * *Token*
)

This function places a completion token into the receive packet queue. This function is always asynchronous.

Parameters:

`[in]` *This* Pointer to the EFI_UDP6_PROTOCOL instance.

`[in]` *Token* Pointer to a token that is associated with the receive data descriptor.

Return values:

EFI_SUCCESS	The receive completion token was cached.
EFI_NOT_STARTED	This EFI UDPv6 Protocol instance has not been started.
EFI_NO_MAPPING	When using a default address, configuration (DHCP, BOOTP, RARP, etc.) is not finished yet.
EFI_INVALID_PARAMETER	One or more of the following conditions is TRUE: This is NULL. Token is NULL. Token.Event is NULL.
EFI_OUT_OF_RESOURCES	The receive completion token could not be queued due to a lack of system resources (usually memory).
EFI_DEVICE_ERROR	

	An unexpected system or network error occurred. The EFI UDPv6 Protocol instance has been reset to startup defaults.
EFI_ACCESS_DENIED	A receive completion token with the same Token.Event is already in the receive queue.
EFI_NOT_READY	The receive request could not be queued because the receive queue is full.

References _UDP6_INSTANCE_DATA::Configured, _UDP6_INSTANCE_DATA::RxTokens, _UDP6_INSTANCE_DATA::TxTokens, UDP6_INSTANCE_DATA_FROM_THIS, Udp6InstanceDeliverDgram(), Udp6ReportIcmpError(), and Udp6TokenExist().

```
EFI_STATUS EFIAPI Udp6Transmit ( IN EFI_UDP6_PROTOCOL *              This,
                                 IN EFI_UDP6_COMPLETION_TOKEN *      Token
                               )
```

This function places a sending request to this instance of the EFI UDPv6 Protocol, alongside the transmit data that was filled by the user.

Parameters:

[in]	*This*	Pointer to the EFI_UDP6_PROTOCOL instance.
[in]	*Token*	Pointer to the completion token that will be placed into the transmit queue.

Return values:

EFI_SUCCESS	The data was queued for transmission.
EFI_NOT_STARTED	This EFI UDPv6 Protocol instance has not been started.
EFI_NO_MAPPING	The under-layer IPv6 driver was responsible for choosing a source address for this instance, but no source address was available for use.
EFI_INVALID_PARAMETER	One or more of the following are TRUE: This is NULL. Token is NULL. Token.Event is NULL. Token.Packet.TxData is NULL. Token.Packet.TxData.FragmentCount is zero. Token.Packet.TxData.DataLength is not equal to the sum of fragment lengths. One or more of the Token.Packet.TxData.FragmentTable[].FragmentLength fields is zero. One or more of the Token.Packet.TxData.FragmentTable[].FragmentBuffer fields is NULL. One or more of the Token.Packet.TxData.UdpSessionData.DestinationAddres are not valid unicast IPv6 addresses if the UdpSessionData is not NULL. Token.Packet.TxData.UdpSessionData. DestinationAddress is NULL Token.Packet.TxData.UdpSessionData. DestinatioPort is zero. Token.Packet.TxData.UdpSessionData is NULL and this instance's UdpConfigData.RemoteAddress is unspecified.
EFI_ACCESS_DENIED	The transmit completion token with the same Token.Event is already in the transmit queue.
EFI_NOT_READY	The completion token could not be queued because the transmit queue is full.
EFI_OUT_OF_RESOURCES	Could not queue the transmit data.
EFI_NOT_FOUND	There is no route to the destination network or address.
EFI_BAD_BUFFER_SIZE	The data length is greater than the maximum UDP packet size. Or, the length of the IP header + UDP header + data length is greater than MTU if DoNotFragment is TRUE.

References _UDP6_INSTANCE_DATA::ConfigData, _UDP6_INSTANCE_DATA::Configured, _UDP6_INSTANCE_DATA::HeadSum, _UDP6_INSTANCE_DATA::IpInfo, _UDP6_SERVICE_DATA::IpIo, _UDP6_INSTANCE_DATA::RxTokens, _UDP6_INSTANCE_DATA::TxTokens, UDP6_HEADER_SIZE, UDP6_INSTANCE_DATA_FROM_THIS, Udp6Checksum(), Udp6NetVectorExtFree(), Udp6RemoveToken(), _UDP6_INSTANCE_DATA::Udp6Service, Udp6TokenExist(), and Udp6ValidateTxToken().

Variable Documentation

EFI_UDP6_PROTOCOL mUdp6Protocol

Initial value:

```
{
 Udp6GetModeData,
 Udp6Configure,
 Udp6Groups,
 Udp6Transmit,
 Udp6Receive,
 Udp6Cancel,
 Udp6Poll
}
```

Referenced by Udp6InitInstance().

NetworkPkg/UefiPxeBcDxe/ComponentName.c File Reference

Functions

EFI_STATUS EFIAPI PxeBcComponentNameGetDriverName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN CHAR8 *Language, OUT CHAR16 **DriverName)

EFI_STATUS EFIAPI PxeBcComponentNameGetControllerName (IN EFI_COMPONENT_NAME_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_HANDLE ChildHandle, IN CHAR8 *Language, OUT CHAR16 **ControllerName)

Variables

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gPxeBcComponentName

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gPxeBcComponentName2

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mPxeBcDriverNameTable []

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mPxeBcControllerNameTable []

Detailed Description

UEFI Component Name(2) protocol implementation for UefiPxeBc driver.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI PxeBcComponentNameGetControllerName (	IN EFI_COMPONENT_NAME_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_HANDLE	*ChildHandle*,
	IN CHAR8 *	*Language*,

OUT CHAR16 ** *ControllerName*
)

Retrieves a Unicode string that is the user-readable name of the controller that is being managed by a driver.

This function retrieves the user-readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

`[in]`	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
`[in]`	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
`[in]`	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
`[in]`	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
`[out]`	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL, and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

Retrieves a Unicode string that is the user-readable name of the controller that is being managed by a driver.

This function retrieves the user-readable name of the controller specified by ControllerHandle and ChildHandle in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the controller name is returned in ControllerName, and EFI_SUCCESS is returned. If the driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle, then EFI_UNSUPPORTED is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.
[in]	*ControllerHandle*	The handle of a controller that the driver specified by This is managing. This handle specifies the controller whose name is to be returned.
[in]	*ChildHandle*	The handle of the child controller to retrieve the name of. This is an optional parameter that may be NULL. It will be NULL for device drivers. It will also be NULL for a bus drivers that wish to retrieve the name of the bus controller. It will not be NULL for a bus driver that wishes to retrieve the name of a child controller.
[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*ControllerName*	A pointer to the Unicode string to return. This Unicode string is the name of the controller specified by ControllerHandle and ChildHandle in the language specified by Language from the point of view of the driver specified by This.

Return values:

EFI_SUCCESS	The Unicode string for the user-readable name in the language specified by Language for the driver specified by This was returned in DriverName.
EFI_INVALID_PARAMETER	ControllerHandle is NULL.
EFI_INVALID_PARAMETER	ChildHandle is not NULL and it is not a valid EFI_HANDLE.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	ControllerName is NULL.
EFI_UNSUPPORTED	The driver specified by This is not currently managing the controller specified by ControllerHandle and ChildHandle.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gPxeBcComponentName, mPxeBcControllerNameTable, PxeBcGetNicByIp4Children(), and PxeBcGetNicByIp6Children().

```
EFI_STATUS EFIAPI
PxeBcComponentNameGetDriverName
  (  IN EFI_COMPONENT_NAME_PROTOCOL *   This,
     IN CHAR8 *                         Language,
     OUT CHAR16 **                      DriverName
  )
```

Retrieves a Unicode string that is the user-readable name of the driver.

This function retrieves the user-readable name of a driver in the form of a Unicode string. If the driver specified by This has a user-readable name in the language specified by Language, then a pointer to the driver name is returned in DriverName, and EFI_SUCCESS is returned. If the driver specified by This does not support the language specified by Language, then EFI_UNSUPPORTED is returned.

Parameters:

[in]	*This*	A pointer to the EFI_COMPONENT_NAME2_PROTOCOL or EFI_COMPONENT_NAME_PROTOCOL instance.

[in]	*Language*	A pointer to a Null-terminated ASCII string array indicating the language. This is the language of the driver name that the caller is requesting, and it must match one of the languages specified in SupportedLanguages. The number of languages supported by a driver is up to the driver writer. Language is specified in RFC 4646 or ISO 639-2 language code format.
[out]	*DriverName*	A pointer to the Unicode string to return. This Unicode string is the name of the driver specified by This in the language specified by Language.

Return values:

EFI_SUCCESS	The Unicode string for the Driver specified by This and the language specified by Language was returned in DriverName.
EFI_INVALID_PARAMETER	Language is NULL.
EFI_INVALID_PARAMETER	DriverName is NULL.
EFI_UNSUPPORTED	The driver specified by This does not support the language specified by Language.

References gPxeBcComponentName, and mPxeBcDriverNameTable.

Variable Documentation

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME_PROTOCOL gPxeBcComponentName

Initial value:

```
{
 PxeBcComponentNameGetDriverName,
 PxeBcComponentNameGetControllerName,
 "eng"
}
```

Referenced by PxeBcComponentNameGetControllerName(), PxeBcComponentNameGetDriverName(), and PxeBcDriverEntryPoint().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_COMPONENT_NAME2_PROTOCOL gPxeBcComponentName2

Initial value:

```
{
 (EFI_COMPONENT_NAME2_GET_DRIVER_NAME) PxeBcComponentNameGetDriverName,
 (EFI_COMPONENT_NAME2_GET_CONTROLLER_NAME) PxeBcComponentNameGetControllerName,
 "en"
}
```

Referenced by PxeBcDriverEntryPoint().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mPxeBcControllerNameTable[]

Initial value:

```
{
 {
   "eng;en",
   L"PXE Controller"
 },
 {
   NULL,
```

```
    NULL
  }
}
```

Referenced by PxeBcComponentNameGetControllerName().

GLOBAL_REMOVE_IF_UNREFERENCED EFI_UNICODE_STRING_TABLE mPxeBcDriverNameTable[]

Initial value:

```
 {
  {
    "eng;en",
    L"UEFI PXE Base Code Driver"
  },
  {
    NULL,
    NULL
  }
}
```

Referenced by PxeBcComponentNameGetDriverName().

Generated on Mon Sep 28 08:49:53 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/UefiPxeBcDxe/PxeBcBoot.c File Reference

Functions

VOID PxeBcDisplayBootItem (IN UINT8 *Str, IN UINT8 Len)
EFI_STATUS PxeBcSelectBootPrompt (IN PXEBC_PRIVATE_DATA *Private)
EFI_STATUS PxeBcSelectBootMenu (IN PXEBC_PRIVATE_DATA *Private, OUT UINT16 *Type, IN BOOLEAN UseDefaultItem)
EFI_STATUS PxeBcDhcp4BootInfo (IN OUT PXEBC_PRIVATE_DATA *Private, OUT UINT64 *BufferSize)
EFI_STATUS PxeBcDhcp6BootInfo (IN OUT PXEBC_PRIVATE_DATA *Private, OUT UINT64 *BufferSize)
EFI_STATUS PxeBcExtractDiscoverInfo (IN PXEBC_PRIVATE_DATA *Private, IN UINT16 Type, IN OUT EFI_PXE_BASE_CODE_DISCOVER_INFO **DiscoverInfo, OUT PXEBC_BOOT_SVR_ENTRY **BootEntry, OUT EFI_PXE_BASE_CODE_SRVLIST **SrvList)
EFI_STATUS PxeBcDiscoverBootServer (IN PXEBC_PRIVATE_DATA *Private, IN UINT16 Type, IN UINT16 *Layer, IN BOOLEAN UseBis, IN EFI_IP_ADDRESS *DestIp, IN UINT16 IpCount, IN EFI_PXE_BASE_CODE_SRVLIST *SrvList)
EFI_STATUS PxeBcDiscoverBootFile (IN OUT PXEBC_PRIVATE_DATA *Private, OUT UINT64 *BufferSize)
EFI_STATUS PxeBcInstallCallback (IN OUT PXEBC_PRIVATE_DATA *Private, OUT BOOLEAN *NewMakeCallback)
VOID PxeBcUninstallCallback (IN PXEBC_PRIVATE_DATA *Private, IN BOOLEAN NewMakeCallback)
EFI_STATUS PxeBcReadBootFileList (IN PXEBC_PRIVATE_DATA *Private, IN OUT UINT64 *BufferSize, IN VOID *Buffer)
EFI_STATUS PxeBcLoadBootFile (IN PXEBC_PRIVATE_DATA *Private, IN OUT UINTN *BufferSize, IN VOID *Buffer)

Detailed Description

Boot functions implementation for UefiPxeBc Driver.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS PxeBcDhcp4BootInfo (IN OUT PXEBC_PRIVATE_DATA * *Private*,

```
                                    OUT UINT64 *                    BufferSize
                                  )
```

Parse out the boot information from the last Dhcp4 reply packet.

Parameters:

[in,out]	*Private*	Pointer to PxeBc private data.
[out]	*BufferSize*	Size of the boot file to be downloaded.

Return values:

EFI_SUCCESS	Successfully parsed out all the boot information.
Others	Failed to parse out the boot information.

References PXEBC_VENDOR_OPTION::BitMap, PXEBC_VENDOR_OPTION::BootSvr, PXEBC_VENDOR_OPTION::BootSvrLen, PXEBC_VENDOR_OPTION::DiscoverCtrl, PXEBC_BOOT_SVR_ENTRY::IpAddr, PXEBC_BOOT_SVR_ENTRY::IpCnt, IS_DISABLE_PROMPT_MENU, IS_VALID_BOOT_SERVERS, PXEBC_DHCP4_PACKET::Offer, PXEBC_DHCP4_PACKET_CACHE::OptList, PXEBC_DHCP4_PACKET_CACHE::Packet, PXEBC_DHCP4_TAG_INDEX_BOOTFILE, PXEBC_DHCP4_TAG_INDEX_BOOTFILE_LEN, PXEBC_DHCP4_TAG_INDEX_SERVER_ID, PxeBcShowIp4Addr(), and PXEBC_DHCP4_PACKET_CACHE::VendorOpt.

Referenced by PxeBcDiscoverBootFile().

```
EFI_STATUS PxeBcDhcp6BootInfo ( IN OUT PXEBC_PRIVATE_DATA *   Private,
                                OUT UINT64 *                  BufferSize
                              )
```

Parse out the boot information from the last Dhcp6 reply packet.

Parameters:

[in,out]	*Private*	Pointer to PxeBc private data.
[out]	*BufferSize*	Size of the boot file to be downloaded.

Return values:

EFI_SUCCESS	Successfully parsed out all the boot information.
EFI_BUFFER_TOO_SMALL	
Others	Failed to parse out the boot information.

References PXEBC_DHCP6_PACKET_CACHE::OptList, PXEBC_DHCP6_IDX_BOOT_FILE_PARAM, PXEBC_DHCP6_IDX_BOOT_FILE_URL, PxeBcExtractBootFileParam(), PxeBcExtractBootFileUrl(), PxeBcSetIp6Address(), and PxeBcShowIp6Addr().

Referenced by PxeBcDiscoverBootFile().

```
EFI_STATUS PxeBcDiscoverBootFile ( IN OUT PXEBC_PRIVATE_DATA *   Private,
                                   OUT UINT64 *                  BufferSize
                                 )
```

Discover all the boot information for boot file.

Parameters:

[in,out]	*Private*	Pointer to PxeBc private data.

[out]	*BufferSize*	Size of the boot file to be downloaded.

Return values:

EFI_SUCCESS	Successfully obtained all the boot information .
EFI_BUFFER_TOO_SMALL	The buffer size is not enough for boot file.
EFI_ABORTED	User cancel current operation.
Others	Failed to parse out the boot information.

References PxeBcDhcp4BootInfo(), PxeBcDhcp6BootInfo(), PxeBcSelectBootMenu(), and PxeBcSelectBootPrompt().

Referenced by PxeBcLoadBootFile().

```
EFI_STATUS PxeBcDiscoverBootServer ( IN PXEBC_PRIVATE_DATA *          Private,
                                     IN UINT16                        Type,
                                     IN UINT16 *                      Layer,
                                     IN BOOLEAN                       UseBis,
                                     IN EFI_IP_ADDRESS *              DestIp,
                                     IN UINT16                        IpCount,
                                     IN EFI_PXE_BASE_CODE_SRVLIST *   SrvList
                                   )
```

Build the discover packet and send out for boot server.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Type*	PxeBc option boot item type.
[in]	*Layer*	Pointer to option boot item layer.
[in]	*UseBis*	Use BIS or not.
[in]	*DestIp*	Pointer to the destination address.
[in]	*IpCount*	The count of the server address.
[in]	*SrvList*	Pointer to the server address list.

Return values:

EFI_SUCCESS	Successfully discovered boot file.
EFI_OUT_OF_RESOURCES	Failed to allocate resource.
EFI_NOT_FOUND	Can't get the PXE reply packet.
Others	Failed to discover boot file.

References PxeBcDhcp4Discover(), and PxeBcDhcp6Discover().

Referenced by EfiPxeBcDiscover().

```
VOID PxeBcDisplayBootItem ( IN UINT8 *  Str,
                            IN UINT8    Len
                          )
```

Display the string of the boot item.

If the length of the boot item string beyond 70 Char, just display 70 Char.

Parameters:

[in]	*Str*	The pointer to the string.
[in]	*Len*	The length of the string.

References PXEBC_DISPLAY_MAX_LINE.

Referenced by PxeBcSelectBootMenu(), and PxeBcSelectBootPrompt().

```
EFI_STATUS
PxeBcExtractDiscoverInfo ( IN PXEBC_PRIVATE_DATA *                  Private,
                           IN UINT16                                Type,
                           IN OUT
                           EFI_PXE_BASE_CODE_DISCOVER_INFO **       DiscoverInfo,
                           OUT PXEBC_BOOT_SVR_ENTRY **              BootEntry,
                           OUT EFI_PXE_BASE_CODE_SRVLIST **         SrvList
                           )
```

Extract the discover information and boot server entry from the cached packets if unspecified.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Type*	The type of bootstrap to perform.
[in,out]	*DiscoverInfo*	Pointer to EFI_PXE_BASE_CODE_DISCOVER_INFO.
[out]	*BootEntry*	Pointer to PXEBC_BOOT_SVR_ENTRY.
[out]	*SrvList*	Pointer to EFI_PXE_BASE_CODE_SRVLIST.

Return values:

EFI_SUCCESS	Successfully extracted the information.
EFI_DEVICE_ERROR	Failed to extract the information.

References PXEBC_VENDOR_OPTION::BitMap, PXEBC_VENDOR_OPTION::BootSvr, PXEBC_VENDOR_OPTION::BootSvrLen, PXEBC_VENDOR_OPTION::DiscoverCtrl, PXEBC_VENDOR_OPTION::DiscoverMcastIp, GET_NEXT_BOOT_SVR_ENTRY, PXEBC_BOOT_SVR_ENTRY::IpAddr, PXEBC_BOOT_SVR_ENTRY::IpCnt, IS_DISABLE_BCAST_DISCOVER, IS_DISABLE_MCAST_DISCOVER, IS_ENABLE_USE_SERVER_LIST, IS_VALID_BOOT_SERVERS, IS_VALID_DISCOVER_VENDOR_OPTION, PXEBC_BOOT_SVR_ENTRY::Type, and PXEBC_DHCP4_PACKET_CACHE::VendorOpt.

Referenced by EfiPxeBcDiscover().

```
EFI_STATUS PxeBcInstallCallback ( IN OUT PXEBC_PRIVATE_DATA *   Private,
                                  OUT BOOLEAN *                 NewMakeCallback
                                  )
```

Install PxeBaseCodeCallbackProtocol if not installed before.

Parameters:

[in,out]	*Private*	Pointer to PxeBc private data.
[out]	*NewMakeCallback*	If TRUE, it is a new callback. Otherwise, it is not new callback.

Return values:

EFI_SUCCESS	PxeBaseCodeCallbackProtocol installed succesfully.
Others	Failed to install PxeBaseCodeCallbackProtocol.

References gPxeBcCallBackTemplate.

Referenced by PxeBcLoadBootFile().

```
EFI_STATUS PxeBcLoadBootFile ( IN PXEBC_PRIVATE_DATA *   Private,
                               IN OUT UINTN *            BufferSize,
                               IN VOID *                 Buffer
                             )
```

Load boot file into user buffer.

Parameters:

`[in]`	*Private*	Pointer to PxeBc private data.
`[in,out]`	*BufferSize*	Size of user buffer for input; required buffer size for output.
`[in]`	*Buffer*	Pointer to user buffer.

Return values:

EFI_SUCCESS	Get all the boot information successfully.
EFI_BUFFER_TOO_SMALL	The buffer size is not enough for boot file.
EFI_ABORTED	User cancelled the current operation.
Others	Failed to parse out the boot information.

References PXEBC_IS_SIZE_OVERFLOWED, PxeBcDiscoverBootFile(), PxeBcInstallCallback(), PxeBcReadBootFileList(), and PxeBcUninstallCallback().

Referenced by EfiPxeLoadFile().

```
EFI_STATUS PxeBcReadBootFileList ( IN PXEBC_PRIVATE_DATA *   Private,
                                   IN OUT UINT64 *           BufferSize,
                                   IN VOID *                 Buffer
                                 )
```

Download one of boot file in the list, and it's special for IPv6.

Parameters:

`[in]`	*Private*	Pointer to PxeBc private data.
`[in,out]`	*BufferSize*	Size of user buffer for input; required buffer size for output.
`[in]`	*Buffer*	Pointer to user buffer.

Return values:

EFI_SUCCESS	Read one of boot file in the list successfully.
EFI_BUFFER_TOO_SMALL	The buffer size is not enough for boot file.
EFI_NOT_FOUND	There is no proper boot file available.
Others	Failed to download boot file in the list.

Referenced by PxeBcLoadBootFile().

```
EFI_STATUS PxeBcSelectBootMenu ( IN PXEBC_PRIVATE_DATA *   Private,
                                 OUT UINT16 *              Type,
                                 IN BOOLEAN                UseDefaultItem
                               )
```

Select the boot menu by user's input.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[out]	*Type*	The type of the menu.
[in]	*UseDefaultItem*	Use default item or not.

Return values:

EFI_ABORTED	User cancel operation.
EFI_SUCCESS	Select the boot menu success.
EFI_NOT_READY	Read the input key from the keybroad has not finish.

References PXEBC_VENDOR_OPTION::BitMap, PXEBC_VENDOR_OPTION::BootMenu, PXEBC_VENDOR_OPTION::BootMenuLen, CTRL, PXEBC_BOOT_MENU_ENTRY::DescLen, PXEBC_DHCP_PACKET_CACHE::Dhcp4, PXEBC_DHCP_PACKET_CACHE::Dhcp6, IS_VALID_BOOT_MENU, PXEBC_DHCP4_PACKET_CACHE::OfferType, PXEBC_DHCP6_PACKET_CACHE::OfferType, PXEBC_DISPLAY_MAX_LINE, PXEBC_MENU_MAX_NUM, PxeBcDisplayBootItem(), PxeOfferTypeDhcpPxe10, PxeOfferTypeProxyPxe10, and PXEBC_DHCP4_PACKET_CACHE::VendorOpt.

Referenced by PxeBcDiscoverBootFile().

```
EFI_STATUS PxeBcSelectBootPrompt ( IN PXEBC_PRIVATE_DATA *   Private   )
```

Select and maintain the boot prompt if needed.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.

Return values:

EFI_SUCCESS	Selected boot prompt done.
EFI_TIMEOUT	Selected boot prompt timed out.
EFI_NOT_FOUND	The proxy offer is not Pxe10.
EFI_ABORTED	User cancelled the operation.
EFI_NOT_READY	Reading the input key from the keyboard has not finish.

References PXEBC_VENDOR_OPTION::BitMap, CTRL, PXEBC_DHCP_PACKET_CACHE::Dhcp4, PXEBC_DHCP_PACKET_CACHE::Dhcp6, PXEBC_VENDOR_OPTION::DiscoverCtrl, IS_DISABLE_PROMPT_MENU, IS_VALID_BOOT_PROMPT, PXEBC_VENDOR_OPTION::MenuPrompt, PXEBC_VENDOR_OPTION::MenuPromptLen, PXEBC_DHCP4_PACKET_CACHE::OfferType, PXEBC_DHCP6_PACKET_CACHE::OfferType, PXEBC_DHCP4_PACKET_CACHE::OptList, PXEBC_MENU_PROMPT::Prompt, PXEBC_DHCP4_TAG_INDEX_BOOTFILE, PxeBcDisplayBootItem(), PxeOfferTypeDhcpPxe10, PxeOfferTypeProxyPxe10, PXEBC_MENU_PROMPT::Timeout, and PXEBC_DHCP4_PACKET_CACHE::VendorOpt.

Referenced by PxeBcDiscoverBootFile().

```
VOID PxeBcUninstallCallback ( IN PXEBC_PRIVATE_DATA *   Private,
                              IN BOOLEAN                NewMakeCallback
                            )
```

Uninstall PxeBaseCodeCallbackProtocol.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*NewMakeCallback*	If TRUE, it is a new callback. Otherwise, it is not new callback.

Referenced by PxeBcLoadBootFile().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/UefiPxeBcDxe/PxeBcBoot.h File Reference

Defines

"memItemRight" valign="bottom">PXEBC_DISPLAY_MAX_LINE 70

#define PXEBC_DEFAULT_UDP_OVERHEAD_S

#define PXEBC_DEFAULT_TFTP_OVERHEAD_

#define PXEBC_IS_SIZE_OVERFLOWED(x) ((s (UINTN) < sizeof (UINT64)) && ((x) > 0xFFFFFFFF))

Functions

EFI_STATUS PxeBcExtractDiscoverInfo (IN PXEBC_PRIVATE_DATA *Private, IN UI Type, IN OUT EFI_PXE_BASE_CODE_DISCOVER_INF **DiscoverInfo, OUT PXEBC_BOOT_SVR_ENTRY **BootEntr EFI_PXE_BASE_CODE_SRVLIST **Srvl

EFI_STATUS PxeBcDiscoverBootServer (IN PXEBC_PRIVATE_DATA *Private, IN UI Type, IN UINT16 *Layer, IN BOOLEAN U IN EFI_IP_ADDRESS *DestIp, IN UINT16 IpCount, IN EFI_PXE_BASE_CODE_SRV *SrvList)

EFI_STATUS PxeBcLoadBootFile (IN PXEBC_PRIVATE_DATA *Private, IN OU UINTN *BufferSize, IN VOID *Buffer)

Detailed Description

Boot functions declaration for UefiPxeBc Driver.

Copyright (c) 2009 - 2012, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by EfiPxeBcStart().

#define PXEBC_DEFAULT_UDP_OVERHEAD_SIZE 8

Referenced by EfiPxeBcStart().

#define PXEBC_DISPLAY_MAX_LINE 70

Referenced by PxeBcDisplayBootItem(), and PxeBcSelectBootMenu().

#define PXEBC_IS_SIZE_OVERFLOWED (x) ((sizeof (UINTN) < sizeof (UINT64)) && ((x) > 0xFFFFFFFF))

Referenced by PxeBcLoadBootFile().

Function Documentation

```
EFI_STATUS PxeBcDiscoverBootServer ( IN PXEBC_PRIVATE_DATA *          Private,
                                     IN UINT16                        Type,
                                     IN UINT16 *                      Layer,
                                     IN BOOLEAN                       UseBis,
                                     IN EFI_IP_ADDRESS *              DestIp,
                                     IN UINT16                        IpCount,
                                     IN EFI_PXE_BASE_CODE_SRVLIST *   SrvList
                                   )
```

Build the discover packet and send out for boot.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Type*	PxeBc option boot item type.
[in]	*Layer*	Pointer to option boot item layer.
[in]	*UseBis*	Use BIS or not.
[in]	*DestIp*	Pointer to the server address.
[in]	*IpCount*	The total count of the server address.
[in]	*SrvList*	Pointer to the server address list.

Return values:

EFI_SUCCESS	Successfully discovered boot file.
EFI_OUT_OF_RESOURCES	Failed to allocate resources.
EFI_NOT_FOUND	Can't get the PXE reply packet.
Others	Failed to discover boot file.

Build the discover packet and send out for boot server.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Type*	PxeBc option boot item type.
[in]	*Layer*	Pointer to option boot item layer.

[in]	*UseBis*	Use BIS or not.
[in]	*DestIp*	Pointer to the destination address.
[in]	*IpCount*	The count of the server address.
[in]	*SrvList*	Pointer to the server address list.

Return values:

EFI_SUCCESS	Successfully discovered boot file.
EFI_OUT_OF_RESOURCES	Failed to allocate resource.
EFI_NOT_FOUND	Can't get the PXE reply packet.
Others	Failed to discover boot file.

References PxeBcDhcp4Discover(), and PxeBcDhcp6Discover().

Referenced by EfiPxeBcDiscover().

```
EFI_STATUS
PxeBcExtractDiscoverInfo ( IN PXEBC_PRIVATE_DATA *                    Private,
                           IN UINT16                                  Type,
                           IN OUT
                           EFI_PXE_BASE_CODE_DISCOVER_INFO **         DiscoverInfo,
                           OUT PXEBC_BOOT_SVR_ENTRY **                BootEntry,
                           OUT EFI_PXE_BASE_CODE_SRVLIST **           SrvList
                           )
```

Extract the discover information and boot server entry from the cached packets if unspecified.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Type*	The type of bootstrap to perform.
[in,out]	*DiscoverInfo*	Pointer to EFI_PXE_BASE_CODE_DISCOVER_INFO.
[out]	*BootEntry*	Pointer to PXEBC_BOOT_SVR_ENTRY.
[out]	*SrvList*	Pointer to EFI_PXE_BASE_CODE_SRVLIST.

Return values:

EFI_SUCCESS	Successfully extracted the information.
EFI_DEVICE_ERROR	Failed to extract the information.

References PXEBC_VENDOR_OPTION::BitMap, PXEBC_VENDOR_OPTION::BootSvr, PXEBC_VENDOR_OPTION::BootSvrLen, PXEBC_VENDOR_OPTION::DiscoverCtrl, PXEBC_VENDOR_OPTION::DiscoverMcastIp, GET_NEXT_BOOT_SVR_ENTRY, PXEBC_BOOT_SVR_ENTRY::IpAddr, PXEBC_BOOT_SVR_ENTRY::IpCnt, IS_DISABLE_BCAST_DISCOVER, IS_DISABLE_MCAST_DISCOVER, IS_ENABLE_USE_SERVER_LIST, IS_VALID_BOOT_SERVERS, IS_VALID_DISCOVER_VENDOR_OPTION, PXEBC_BOOT_SVR_ENTRY::Type, and PXEBC_DHCP4_PACKET_CACHE::VendorOpt.

Referenced by EfiPxeBcDiscover().

```
EFI_STATUS PxeBcLoadBootFile ( IN PXEBC_PRIVATE_DATA *    Private,
                               IN OUT UINTN *             BufferSize,
```

IN VOID * *Buffer*
)

Load boot file into user buffer.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in,out]	*BufferSize*	Size of user buffer for input; required buffer size for output.
[in]	*Buffer*	Pointer to user buffer.

Return values:

EFI_SUCCESS	Successfully obtained all the boot information.
EFI_BUFFER_TOO_SMALL	The buffer size is not enough for boot file.
EFI_ABORTED	User cancelled the current operation.
Others	Failed to parse out the boot information.

Load boot file into user buffer.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in,out]	*BufferSize*	Size of user buffer for input; required buffer size for output.
[in]	*Buffer*	Pointer to user buffer.

Return values:

EFI_SUCCESS	Get all the boot information successfully.
EFI_BUFFER_TOO_SMALL	The buffer size is not enough for boot file.
EFI_ABORTED	User cancelled the current operation.
Others	Failed to parse out the boot information.

References PXEBC_IS_SIZE_OVERFLOWED, PxeBcDiscoverBootFile(), PxeBcInstallCallback(), PxeBcReadBootFileList(), and PxeBcUninstallCallback().

Referenced by EfiPxeLoadFile().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/UefiPxeBcDxe/PxeBcDhcp4.c File Reference

Functions

EFI_DHCP4_PACKET_OPTION * PxeBcParseDhcp4Options (IN UINT8 *Buffer, IN UINT32 Length, IN UINT8 OptTag)
VOID PxeBcParseVendorOptions (IN EFI_DHCP4_PACKET_OPTION *Dhcp4Option, IN PXEBC_VENDOR_OPTION *VendorOption)
UINT32 PxeBcBuildDhcp4Options (IN PXEBC_PRIVATE_DATA *Private, OUT EFI_DHCP4_PACKET_OPTION **OptList, IN UINT8 *Buffer, IN BOOLEAN NeedMsgType)
VOID PxeBcSeedDhcp4Packet (OUT EFI_DHCP4_PACKET *Seed, IN EFI_UDP4_PROTOCOL *Udp4)
VOID PxeBcCacheDhcp4Packet (IN EFI_DHCP4_PACKET *Dst, IN EFI_DHCP4_PACKET *Src)
EFI_STATUS PxeBcParseDhcp4Packet (IN PXEBC_DHCP4_PACKET_CACHE *Cache4)
VOID PxeBcCopyDhcp4Ack (IN PXEBC_PRIVATE_DATA *Private, IN EFI_DHCP4_PACKET *Ack, IN BOOLEAN Verified)
VOID PxeBcCopyProxyOffer (IN PXEBC_PRIVATE_DATA *Private, IN UINT32 OfferIndex)
EFI_STATUS PxeBcRetryBinlOffer (IN PXEBC_PRIVATE_DATA *Private, IN UINT32 Index)
VOID PxeBcCacheDhcp4Offer (IN PXEBC_PRIVATE_DATA *Private, IN EFI_DHCP4_PACKET *RcvdOffer)
VOID PxeBcSelectDhcp4Offer (IN PXEBC_PRIVATE_DATA *Private)
EFI_STATUS PxeBcHandleDhcp4Offer (IN PXEBC_PRIVATE_DATA *Private)
EFI_STATUS EFIAPI PxeBcDhcp4CallBack (IN EFI_DHCP4_PROTOCOL *This, IN VOID *Context, IN EFI_DHCP4_STATE CurrentState, IN EFI_DHCP4_EVENT Dhcp4Event, IN EFI_DHCP4_PACKET *Packet, OUT EFI_DHCP4_PACKET **NewPacket)
EFI_STATUS PxeBcDhcp4Discover (IN PXEBC_PRIVATE_DATA *Private, IN UINT16 Type, IN UINT16 *Layer, IN BOOLEAN UseBis, IN EFI_IP_ADDRESS *DestIp, IN UINT16 IpCount, IN EFI_PXE_BASE_CODE_SRVLIST *SrvList)
EFI_STATUS PxeBcSetIp4Policy (IN PXEBC_PRIVATE_DATA *Private)
EFI_STATUS PxeBcDhcp4Dora (IN PXEBC_PRIVATE_DATA *Private, IN EFI_DHCP4_PROTOCOL *Dhcp4)

Variables

UINT8 mInterestedDhcp4Tags [PXEBC_DHCP4_TAG_INDEX_MAX]
UINT32 mPxeDhcpTimeout [4] = {4, 8, 16, 32}

Detailed Description

Functions implementation related with DHCPv4 for UefiPxeBc Driver.

Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
UINT32 PxeBcBuildDhcp4Options ( IN PXEBC_PRIVATE_DATA *              Private,
                                OUT EFI_DHCP4_PACKET_OPTION **       OptList,
                                IN UINT8 *                           Buffer,
                                IN BOOLEAN                           NeedMsgType
                              )
```

Build the options buffer for the DHCPv4 request packet.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[out]	*OptList*	Pointer to the option pointer array.
[in]	*Buffer*	Pointer to the buffer to contain the option list.
[in]	*NeedMsgType*	If TRUE, it is necessary to include the Msg type option. Otherwise, it is not necessary.

Returns:

Index The count of the built-in options.

References PXEBC_DHCP4_OPTION_ENTRY::Arch, PXEBC_DHCP4_OPTION_CLID::ArchitectureType, PXEBC_DHCP4_OPTION_ENTRY::Clid, DEFAULT_CLASS_ID_DATA, DEFAULT_UNDI_MAJOR, DEFAULT_UNDI_MINOR, DEFAULT_UNDI_TYPE, GET_NEXT_DHCP_OPTION, PXEBC_DHCP4_OPTION_UUID::Guid, PXEBC_DHCP4_OPTION_CLID::InterfaceName, PXEBC_DHCP4_OPTION_UNDI::MajorVer, PXEBC_DHCP4_OPTION_ENTRY::MaxMesgSize, PXEBC_DHCP4_OPTION_ENTRY::Mesg, PXEBC_DHCP4_OPTION_UNDI::MinorVer, PXEBC_DHCP4_OPTION_ENTRY::Para, PXEBC_DHCP4_OPTION_PARA::ParaList, PXEBC_DHCP4_MSG_TYPE_REQUEST, PXEBC_DHCP4_PACKET_MAX_SIZE, PXEBC_DHCP4_TAG_BOOTFILE, PXEBC_DHCP4_TAG_BOOTFILE_LEN, PXEBC_DHCP4_TAG_BROADCAST, PXEBC_DHCP4_TAG_CLASS_ID, PXEBC_DHCP4_TAG_DNS_SERVER, PXEBC_DHCP4_TAG_DOMAINNAME, PXEBC_DHCP4_TAG_EMTU, PXEBC_DHCP4_TAG_EXTEND_PATH, PXEBC_DHCP4_TAG_HOSTNAME, PXEBC_DHCP4_TAG_LEASE, PXEBC_DHCP4_TAG_MAXMSG, PXEBC_DHCP4_TAG_MSG_TYPE, PXEBC_DHCP4_TAG_NAME_SERVER, PXEBC_DHCP4_TAG_NETMASK, PXEBC_DHCP4_TAG_NIS_DOMAIN, PXEBC_DHCP4_TAG_NIS_SERVER, PXEBC_DHCP4_TAG_NTP_SERVER, PXEBC_DHCP4_TAG_PARA_LIST, PXEBC_DHCP4_TAG_REQUEST_IP, PXEBC_DHCP4_TAG_ROOTPATH, PXEBC_DHCP4_TAG_ROUTER, PXEBC_DHCP4_TAG_SERVER_ID, PXEBC_DHCP4_TAG_T1, PXEBC_DHCP4_TAG_T2, PXEBC_DHCP4_TAG_TFTP, PXEBC_DHCP4_TAG_TIME_OFFSET, PXEBC_DHCP4_TAG_TIME_SERVER, PXEBC_DHCP4_TAG_TTL, PXEBC_DHCP4_TAG_VENDOR, PXEBC_PXE_DHCP4_TAG_ARCH, PXEBC_PXE_DHCP4_TAG_UNDI, PXEBC_PXE_DHCP4_TAG_UUID, PxeBcUintnToAscDecWithFormat(),

PXEBC_DHCP4_OPTION_MAX_MESG_SIZE::Size, PXEBC_DHCP4_OPTION_ARCH::Type, PXEBC_DHCP4_OPTION_UNDI::Type, PXEBC_DHCP4_OPTION_UUID::Type, PXEBC_DHCP4_OPTION_MESG::Type, PXEBC_DHCP4_OPTION_ENTRY::Undi, PXEBC_DHCP4_OPTION_CLID::UndiMajor, PXEBC_DHCP4_OPTION_CLID::UndiMinor, and PXEBC_DHCP4_OPTION_ENTRY::Uuid.

Referenced by PxeBcDhcp4Discover(), and PxeBcDhcp4Dora().

```
VOID PxeBcCacheDhcp4Offer ( IN PXEBC_PRIVATE_DATA *   Private,
                            IN EFI_DHCP4_PACKET *     RcvdOffer
                          )
```

Cache all the received DHCPv4 offers, and set OfferIndex and OfferCount.

Parameters:

[in] *Private* Pointer to PxeBc private data.
[in] *RcvdOffer* Pointer to the received offer packet.

References IS_PROXY_DHCP_OFFER, PXEBC_DHCP4_PACKET::Offer, PXEBC_DHCP4_PACKET_CACHE::OfferType, PXEBC_DHCP4_PACKET_CACHE::Packet, PXEBC_OFFER_MAX_NUM, PxeBcCacheDhcp4Packet(), PxeBcParseDhcp4Packet(), PxeOfferTypeBootp, PxeOfferTypeMax, and PxeOfferTypeProxyBinl.

Referenced by PxeBcDhcp4CallBack().

```
VOID PxeBcCacheDhcp4Packet ( IN EFI_DHCP4_PACKET *   Dst,
                             IN EFI_DHCP4_PACKET *   Src
                           )
```

Cache the DHCPv4 packet.

Parameters:

[in] *Dst* Pointer to the cache buffer for DHCPv4 packet.
[in] *Src* Pointer to the DHCPv4 packet to be cached.

Referenced by PxeBcCacheDhcp4Offer(), PxeBcCopyDhcp4Ack(), PxeBcCopyProxyOffer(), and PxeBcDhcp4Discover().

```
VOID PxeBcCopyDhcp4Ack ( IN PXEBC_PRIVATE_DATA *   Private,
                         IN EFI_DHCP4_PACKET *     Ack,
                         IN BOOLEAN                Verified
                       )
```

Cache the DHCPv4 ack packet, and parse it on demand.

Parameters:

[in] *Private* Pointer to PxeBc private data.
[in] *Ack* Pointer to the DHCPv4 ack packet.

[in] *Verified* If TRUE, parse the ACK packet and store info into mode data.

References PxeBcCacheDhcp4Packet(), and PxeBcParseDhcp4Packet().

Referenced by PxeBcDhcp4CallBack(), and PxeBcHandleDhcp4Offer().

```
VOID PxeBcCopyProxyOffer ( IN PXEBC_PRIVATE_DATA * Private,
                           IN UINT32               OfferIndex
                         )
```

Cache the DHCPv4 proxy offer packet according to the received order.

Parameters:

[in] *Private* Pointer to PxeBc private data.
[in] *OfferIndex* The received order of offer packets.

References PXEBC_OFFER_MAX_NUM, PxeBcCacheDhcp4Packet(), and PxeBcParseDhcp4Packet().

Referenced by PxeBcHandleDhcp4Offer().

```
EFI_STATUS EFIAPI PxeBcDhcp4CallBack ( IN EFI_DHCP4_PROTOCOL *  This,
                                       IN VOID *                Context,
                                       IN EFI_DHCP4_STATE       CurrentState,
                                       IN EFI_DHCP4_EVENT       Dhcp4Event,
                                       IN EFI_DHCP4_PACKET *    Packet,
                                       OUT EFI_DHCP4_PACKET **  NewPacket
                                     )
```

EFI_DHCP4_CALLBACK is provided by the consumer of the EFI DHCPv4 Protocol driver to intercept events that occurred in the configuration process.

Parameters:

[in] *This* Pointer to the EFI DHCPv4 Protocol.
[in] *Context* Pointer to the context set by EFI_DHCP4_PROTOCOL.Configure().
[in] *CurrentState* The current operational state of the EFI DHCPv4 Protocol driver.
[in] *Dhcp4Event* The event that occurs in the current state, which usually means a state transition.
[in] *Packet* The DHCPv4 packet that is going to be sent or already received.
[out] *NewPacket* The packet that is used to replace the above Packet.

Return values:

EFI_SUCCESS Tells the EFI DHCPv4 Protocol driver to continue the DHCP process.
EFI_NOT_READY Only used in the Dhcp4Selecting state. The EFI DHCPv4 Protocol driver will continue to wait for more DHCPOFFER packets until the retry timeout expires.
EFI_ABORTED Tells the EFI DHCPv4 Protocol driver to abort the current process and return to the Dhcp4Init or Dhcp4InitReboot state.

References PXEBC_DHCP_PACKET_CACHE::Dhcp4, _PXEBC_PRIVATE_DATA::Function, GET_OPTION_BUFFER_LEN, PXEBC_DHCP4_PACKET::Offer,

_PXEBC_PRIVATE_DATA::OfferBuffer, _PXEBC_PRIVATE_DATA::OfferNum, PXEBC_DHCP4_PACKET_CACHE::Packet, _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_DHCP4_PACKET_MAX_SIZE, PXEBC_DHCP4_TAG_MAXMSG, PXEBC_OFFER_MAX_NUM, PxeBcCacheDhcp4Offer(), _PXEBC_PRIVATE_DATA::PxeBcCallback, PxeBcCopyDhcp4Ack(), PxeBcParseDhcp4Options(), PxeBcSelectDhcp4Offer(), and _PXEBC_PRIVATE_DATA::SelectIndex.

Referenced by PxeBcDhcp4Dora().

```
EFI_STATUS PxeBcDhcp4Discover ( IN PXEBC_PRIVATE_DATA *          Private,
                                IN UINT16                        Type,
                                IN UINT16 *                      Layer,
                                IN BOOLEAN                       UseBis,
                                IN EFI_IP_ADDRESS *              DestIp,
                                IN UINT16                        IpCount,
                                IN EFI_PXE_BASE_CODE_SRVLIST *   SrvList
                              )
```

Build and send out the request packet for the bootfile, and parse the reply.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Type*	PxeBc option boot item type.
[in]	*Layer*	Pointer to option boot item layer.
[in]	*UseBis*	Use BIS or not.
[in]	*DestIp*	Pointer to the server address.
[in]	*IpCount*	The total count of the server address.
[in]	*SrvList*	Pointer to EFI_PXE_BASE_CODE_SRVLIST.

Return values:

EFI_SUCCESS	Successfully discovered boot file.
EFI_OUT_OF_RESOURCES	Failed to allocate resource.
EFI_NOT_FOUND	Can't get the PXE reply packet.
Others	Failed to discover boot file.

References PXEBC_OPTION_BOOT_ITEM::Layer, PXEBC_BOOT_REQUEST_RETRIES, PXEBC_BOOT_REQUEST_TIMEOUT, PXEBC_BS_DISCOVER_PORT, PXEBC_DHCP4_OPTION_MAX_NUM, PXEBC_DHCP4_OPTION_MAX_SIZE, PXEBC_DHCP4_S_PORT, PXEBC_DHCP4_TAG_EOP, PXEBC_DHCP4_TAG_VENDOR, PXEBC_VENDOR_TAG_BOOT_ITEM, PxeBcBuildDhcp4Options(), PxeBcCacheDhcp4Packet(), and PXEBC_OPTION_BOOT_ITEM::Type.

Referenced by PxeBcDiscoverBootServer(), and PxeBcRetryBinlOffer().

```
EFI_STATUS PxeBcDhcp4Dora ( IN PXEBC_PRIVATE_DATA *   Private,
                            IN EFI_DHCP4_PROTOCOL *   Dhcp4
                          )
```

Start the D.O.R.A DHCPv4 process to acquire the IPv4 address and other PXE boot information.

Parameters:

[in] *Private* Pointer to PxeBc private data.
[in] *Dhcp4* Pointer to the EFI_DHCP4_PROTOCOL

Return values:

EFI_SUCCESS The D.O.R.A process successfully finished.
Others Failed to finish the D.O.R.A process.

References mPxeDhcpTimeout, PXEBC_DHCP4_OPTION_MAX_NUM, PXEBC_DHCP4_OPTION_MAX_SIZE, PXEBC_DHCP_RETRIES, PxeBcBuildDhcp4Options(), PxeBcDhcp4CallBack(), PxeBcFlushStationIp(), PxeBcHandleDhcp4Offer(), and PxeBcShowIp4Addr().

Referenced by EfiPxeBcDhcp().

```
EFI_STATUS PxeBcHandleDhcp4Offer ( IN PXEBC_PRIVATE_DATA * Private )
```

Handle the DHCPv4 offer packet.

Parameters:

[in] *Private* Pointer to PxeBc private data.

Return values:

EFI_SUCCESS Handled the DHCPv4 offer packet successfully.
EFI_NO_RESPONSE No response to the following request packet.
EFI_NOT_FOUND No boot filename received.

References IS_PROXY_DHCP_OFFER, PXEBC_DHCP4_PACKET::Offer, PXEBC_DHCP4_PACKET_CACHE::OfferType, PXEBC_DHCP4_PACKET_CACHE::OptList, PXEBC_DHCP4_PACKET_CACHE::Packet, PXEBC_DHCP4_TAG_INDEX_BOOTFILE, PXEBC_OFFER_MAX_NUM, PxeBcCopyDhcp4Ack(), PxeBcCopyProxyOffer(), PxeBcRetryBinlOffer(), PxeOfferTypeBootp, PxeOfferTypeDhcpBinl, PxeOfferTypeDhcpOnly, PxeOfferTypeMax, and PxeOfferTypeProxyBinl.

Referenced by PxeBcDhcp4Dora().

```
EFI_DHCP4_PACKET_OPTION* PxeBcParseDhcp4Options ( IN UINT8 * Buffer,
                                                  IN UINT32 Length,
                                                  IN UINT8 OptTag
                                                )
```

Parse a certain dhcp4 option by OptTag in Buffer, and return with start pointer.

Parameters:

[in] *Buffer* Pointer to the option buffer.
[in] *Length* Length of the option buffer.
[in] *OptTag* Tag of the required option.

Return values:

NULL Failed to find the required option.
Others The position of the required option.

References PXEBC_DHCP4_TAG_EOP, and PXEBC_DHCP4_TAG_PAD.

Referenced by PxeBcDhcp4CallBack(), and PxeBcParseDhcp4Packet().

EFI_STATUS PxeBcParseDhcp4Packet (IN PXEBC_DHCP4_PACKET_CACHE * *Cache4*)

Parse the cached DHCPv4 packet, including all the options.

Parameters:

[in] *Cache4* Pointer to cached DHCPv4 packet.

Return values:

EFI_SUCCESS	Parsed the DHCPv4 packet successfully.
EFI_DEVICE_ERROR	Failed to parse and invalid packet.

References DEFAULT_CLASS_ID_DATA, GET_OPTION_BUFFER_LEN, IS_VALID_DISCOVER_VENDOR_OPTION, IS_VALID_MTFTP_VENDOR_OPTION, mInterestedDhcp4Tags, PXEBC_DHCP4_OVERLOAD_FILE, PXEBC_DHCP4_OVERLOAD_SERVER_NAME, PXEBC_DHCP4_TAG_INDEX_BOOTFILE, PXEBC_DHCP4_TAG_INDEX_CLASS_ID, PXEBC_DHCP4_TAG_INDEX_MAX, PXEBC_DHCP4_TAG_INDEX_MSG_TYPE, PXEBC_DHCP4_TAG_INDEX_OVERLOAD, PXEBC_DHCP4_TAG_INDEX_VENDOR, PxeBcParseDhcp4Options(), PxeBcParseVendorOptions(), PxeOfferTypeBootp, PxeOfferTypeDhcpBinl, PxeOfferTypeDhcpOnly, PxeOfferTypeDhcpPxe10, PxeOfferTypeProxyBinl, and PxeOfferTypeProxyPxe10.

Referenced by EfiPxeBcDiscover(), PxeBcCacheDhcp4Offer(), PxeBcCopyDhcp4Ack(), PxeBcCopyProxyOffer(), and PxeBcRetryBinlOffer().

VOID PxeBcParseVendorOptions (IN EFI_DHCP4_PACKET_OPTION * *Dhcp4Option*,
IN PXEBC_VENDOR_OPTION * *VendorOption*
)

Parse the PXE vender options and extract the information from them.

Parameters:

[in] *Dhcp4Option* Pointer to vendor options in buffer.

[in] *VendorOption* Pointer to structure to store information in vendor options.

References PXEBC_DHCP4_TAG_EOP, PXEBC_DHCP4_TAG_PAD, PXEBC_VENDOR_TAG_BOOT_ITEM, PXEBC_VENDOR_TAG_BOOT_MENU, PXEBC_VENDOR_TAG_BOOT_SERVERS, PXEBC_VENDOR_TAG_CREDENTIAL_TYPES, PXEBC_VENDOR_TAG_DISCOVER_CTRL, PXEBC_VENDOR_TAG_DISCOVER_MCAST, PXEBC_VENDOR_TAG_MCAST_ALLOC, PXEBC_VENDOR_TAG_MENU_PROMPT, PXEBC_VENDOR_TAG_MTFTP_CPORT, PXEBC_VENDOR_TAG_MTFTP_DELAY, PXEBC_VENDOR_TAG_MTFTP_IP, PXEBC_VENDOR_TAG_MTFTP_SPORT, PXEBC_VENDOR_TAG_MTFTP_TIMEOUT, and SET_VENDOR_OPTION_BIT_MAP.

Referenced by PxeBcParseDhcp4Packet().

EFI_STATUS PxeBcRetryBinlOffer (IN PXEBC_PRIVATE_DATA * *Private*,
IN UINT32 *Index*
)

Retry to request bootfile name by the BINL offer.

Parameters:

[in] *Private* Pointer to PxeBc private data.
[in] *Index* The received order of offer packets.

Return values:

EFI_SUCCESS Successfully retried to request bootfile name.
EFI_DEVICE_ERROR Failed to retry bootfile name.

References PXEBC_DHCP4_PACKET::Offer, PXEBC_DHCP4_PACKET_CACHE::OfferType, PXEBC_DHCP4_PACKET_CACHE::OptList, PXEBC_DHCP4_PACKET_CACHE::Packet, PXEBC_DHCP4_TAG_INDEX_BOOTFILE, PXEBC_DHCP4_TAG_INDEX_SERVER_ID, PXEBC_OFFER_MAX_NUM, PxeBcDhcp4Discover(), PxeBcParseDhcp4Packet(), PxeOfferTypeDhcpBinl, PxeOfferTypeProxyBinl, PxeOfferTypeProxyPxe10, and PxeOfferTypeProxyWfm11a.

Referenced by PxeBcHandleDhcp4Offer().

```
VOID PxeBcSeedDhcp4Packet ( OUT EFI_DHCP4_PACKET *   Seed,
                            IN EFI_UDP4_PROTOCOL *   Udp4
                          )
```

Create a template DHCPv4 packet as a seed.

Parameters:

[out] *Seed* Pointer to the seed packet.
[in] *Udp4* Pointer to EFI_UDP4_PROTOCOL.

References PXEBC_DHCP4_MAGIC, PXEBC_DHCP4_OPCODE_REQUEST, and PXEBC_DHCP4_TAG_EOP.

Referenced by EfiPxeBcStart().

```
VOID PxeBcSelectDhcp4Offer ( IN PXEBC_PRIVATE_DATA *  Private )
```

Select an DHCPv4 offer, and record SelectIndex and SelectProxyType.

Parameters:

[in] *Private* Pointer to PxeBc private data.

References IS_PROXY_DHCP_OFFER, PXEBC_DHCP4_TAG_INDEX_BOOTFILE, PxeOfferTypeBootp, PxeOfferTypeDhcpBinl, PxeOfferTypeDhcpOnly, PxeOfferTypeDhcpPxe10, PxeOfferTypeDhcpWfm11a, PxeOfferTypeProxyBinl, PxeOfferTypeProxyPxe10, and PxeOfferTypeProxyWfm11a.

Referenced by PxeBcDhcp4CallBack().

```
EFI_STATUS PxeBcSetIp4Policy ( IN PXEBC_PRIVATE_DATA *  Private )
```

Switch the Ip4 policy to static.

Parameters:

[in] *Private* The pointer to PXEBC_PRIVATE_DATA.

Return values:

EFI_SUCCESS The policy is already configured to static.
Others Other error as indicated..

References Policy.

Referenced by EfiPxeBcStart().

Variable Documentation

UINT8 mInterestedDhcp4Tags[PXEBC_DHCP4_TAG_INDEX_MAX]

Initial value:

```
{
 PXEBC_DHCP4_TAG_BOOTFILE_LEN,
 PXEBC_DHCP4_TAG_VENDOR,
 PXEBC_DHCP4_TAG_OVERLOAD,
 PXEBC_DHCP4_TAG_MSG_TYPE,
 PXEBC_DHCP4_TAG_SERVER_ID,
 PXEBC_DHCP4_TAG_CLASS_ID,
 PXEBC_DHCP4_TAG_BOOTFILE
}
```

UINT32 mPxeDhcpTimeout[4] = {4, 8, 16, 32}

Referenced by PxeBcDhcp4Dora().

NetworkPkg/UefiPxeBcDxe/PxeBcDhcp4.h File Reference

Data Structures

struct PXEBC_DHCP4_OPTION_PARA
struct PXEBC_DHCP4_OPTION_MAX_
struct PXEBC_DHCP4_OPTION_UNDI
struct PXEBC_DHCP4_OPTION_MESG
struct PXEBC_DHCP4_OPTION_ARCH
struct PXEBC_DHCP4_OPTION_CLID
struct PXEBC_DHCP4_OPTION_UUID
struct PXEBC_OPTION_BOOT_ITEM
union PXEBC_DHCP4_OPTION_ENTR
struct PXEBC_BOOT_SVR_ENTRY
struct PXEBC_BOOT_MENU_ENTRY
struct PXEBC_MENU_PROMPT
struct PXEBC_VENDOR_OPTION
union PXEBC_DHCP4_PACKET
struct PXEBC_DHCP4_PACKET_CACH

Defines

"memItemRight" valign="bottom">PXEBC_DHCP4_OPTION_MAX_NUM 16
#define PXEBC_DHCP4_OPTION_MAX_
#define PXEBC_DHCP4_PACKET_MAX_
#define PXEBC_DHCP4_S_PORT 67
#define PXEBC_DHCP4_C_PORT 68
#define PXEBC_BS_DOWNLOAD_PORT
#define PXEBC_BS_DISCOVER_PORT
#define PXEBC_DHCP4_OPCODE_REQU
#define PXEBC_DHCP4_OPCODE_REPL
#define PXEBC_DHCP4_MSG_TYPE_RE
#define PXEBC_DHCP4_MAGIC 0x6353
#define PXEBC_DHCP4_TAG_PAD 0
#define PXEBC_DHCP4_TAG_EOP 255
#define PXEBC_DHCP4_TAG_NETMASK
#define PXEBC_DHCP4_TAG_TIME_OF
#define PXEBC_DHCP4_TAG_ROUTER
#define PXEBC_DHCP4_TAG_TIME_SE
#define PXEBC_DHCP4_TAG_NAME_SE
#define PXEBC_DHCP4_TAG_DNS_SER
#define PXEBC_DHCP4_TAG_HOSTNAM
#define PXEBC_DHCP4_TAG_BOOTFIL
#define PXEBC_DHCP4_TAG_DUMP 14
#define PXEBC_DHCP4_TAG_DOMAINN
#define PXEBC_DHCP4_TAG_ROOTPAT
#define PXEBC_DHCP4_TAG_EXTEND_
#define PXEBC_DHCP4_TAG_EMTU 22
#define PXEBC_DHCP4_TAG_TTL 23
#define PXEBC_DHCP4_TAG_BROADC

```
#define PXEBC_DHCP4_TAG_NIS_DOMAIN  40
#define PXEBC_DHCP4_TAG_NIS_SERVER  41
#define PXEBC_DHCP4_TAG_NTP_SERVER  42
#define PXEBC_DHCP4_TAG_VENDOR  43
#define PXEBC_DHCP4_TAG_REQUEST_IP  50
#define PXEBC_DHCP4_TAG_LEASE  51
#define PXEBC_DHCP4_TAG_OVERLOAD  52
#define PXEBC_DHCP4_TAG_MSG_TYPE  53
#define PXEBC_DHCP4_TAG_SERVER_ID  54
#define PXEBC_DHCP4_TAG_PARA_LIST  55
#define PXEBC_DHCP4_TAG_MAXMSG  57
#define PXEBC_DHCP4_TAG_T1  58
#define PXEBC_DHCP4_TAG_T2  59
#define PXEBC_DHCP4_TAG_CLASS_ID  60
#define PXEBC_DHCP4_TAG_CLIENT_ID  61
#define PXEBC_DHCP4_TAG_TFTP  66
#define PXEBC_DHCP4_TAG_BOOTFILE  67
#define PXEBC_PXE_DHCP4_TAG_ARCH  93
#define PXEBC_PXE_DHCP4_TAG_UNDI  94
#define PXEBC_PXE_DHCP4_TAG_UUID  97
#define PXEBC_VENDOR_TAG_MTFTP_IP  1
#define PXEBC_VENDOR_TAG_MTFTP_CPORT
#define PXEBC_VENDOR_TAG_MTFTP_SPORT
#define PXEBC_VENDOR_TAG_MTFTP_TIMEO
#define PXEBC_VENDOR_TAG_MTFTP_DELAY
#define PXEBC_VENDOR_TAG_DISCOVER_CT
#define PXEBC_VENDOR_TAG_DISCOVER_MC
#define PXEBC_VENDOR_TAG_BOOT_SERVER
#define PXEBC_VENDOR_TAG_BOOT_MENU
#define PXEBC_VENDOR_TAG_MENU_PROMP
#define PXEBC_VENDOR_TAG_MCAST_ALLO
#define PXEBC_VENDOR_TAG_CREDENTIAL_
#define PXEBC_VENDOR_TAG_BOOT_ITEM  7
#define PXEBC_BOOT_REQUEST_TIMEOUT  1
#define PXEBC_BOOT_REQUEST_RETRIES  4
#define PXEBC_DHCP4_OVERLOAD_FILE  1
#define PXEBC_DHCP4_OVERLOAD_SERVER_
#define PXEBC_DHCP4_TAG_INDEX_BOOTFIL
#define PXEBC_DHCP4_TAG_INDEX_VENDOR
#define PXEBC_DHCP4_TAG_INDEX_OVERLO
#define PXEBC_DHCP4_TAG_INDEX_MSG_TY
#define PXEBC_DHCP4_TAG_INDEX_SERVER_
#define PXEBC_DHCP4_TAG_INDEX_CLASS_I
#define PXEBC_DHCP4_TAG_INDEX_BOOTFIL
#define PXEBC_DHCP4_TAG_INDEX_MAX  7
#define BIT(x)  (1 << x)
#define CTRL(x)  (0x1F & (x))
#define DEFAULT_CLASS_ID_DATA  "PXEClie
#define DEFAULT_UNDI_TYPE  1
#define DEFAULT_UNDI_MAJOR  3
#define DEFAULT_UNDI_MINOR  0
#define MTFTP_VENDOR_OPTION_BIT_MAP
#define DISCOVER_VENDOR_OPTION_BIT_MA
#define IS_VALID_BOOT_SERVERS(x)
```

#define IS_VALID_BOOT_PROMPT(x)
#define IS_VALID_BOOT_MENU(x)
#define IS_VALID_MTFTP_VENDOR_O
#define IS_VALID_DISCOVER_VENDO
DISCOVER_VENDOR_OPTION_
#define IS_VALID_CREDENTIAL_VEND
#define IS_VALID_BOOTITEM_VENDO
#define SET_VENDOR_OPTION_BIT_M
((UINT32) ((x)[(y) / 32]) | BIT ((y)
#define GET_NEXT_DHCP_OPTION(Opt
#define GET_OPTION_BUFFER_LEN(Pk
(EFI_DHCP4_HEADER) - 4)
#define GET_NEXT_BOOT_SVR_ENTRY
#define IS_PROXY_DHCP_OFFER(Offer)
(&(Offer)->Dhcp4.Header.YourAd
#define IS_DISABLE_BCAST_DISCOVE
#define IS_DISABLE_MCAST_DISCOVE
#define IS_ENABLE_USE_SERVER_LIS
#define IS_DISABLE_PROMPT_MENU(x

Enumerations

enum PXEBC_OFFER_TYPE {
PxeOfferTypeDhcpOnly, PxeOffe
PxeOfferTypeDhcpWfm11a, PxeOf
PxeOfferTypeProxyPxe10, PxeOff
PxeOfferTypeProxyBinl, PxeOfferT
PxeOfferTypeMax
}

Functions

VOID PxeBcSeedDhcp4Packet (OUT EFI
EFI_UDP4_PROTOCOL *Udp4)
EFI_STATUS PxeBcParseDhcp4Packet (IN PXE
*Cache4)
EFI_STATUS PxeBcDhcp4Discover (IN PXEBC_
UINT16 Type, IN UINT16 *Layer,
EFI_IP_ADDRESS *DestIp, IN U
EFI_PXE_BASE_CODE_SRVLIS
EFI_STATUS PxeBcSetIp4Policy (IN PXEBC_P
EFI_STATUS PxeBcDhcp4Dora (IN PXEBC_PR
EFI_DHCP4_PROTOCOL *Dhcp4

Detailed Description

Functions declaration related with DHCPv4 for UefiPxeBc Driver.

Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at

http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"paramtype">x) (1 << x)

#define CTRL (x) (0x1F & (x))

Referenced by PxeBcSelectBootMenu(), and PxeBcSelectBootPrompt().

#define DEFAULT_CLASS_ID_DATA "PXEClient:Arch:xxxxx:UNDI:003000"

#define DEFAULT_UNDI_MAJOR 3

#define DEFAULT_UNDI_MINOR 0

#define DEFAULT_UNDI_TYPE 1

#define DISCOVER_VENDOR_OPTION_BIT_MAP

Value:

```
(BIT (PXEBC_VENDOR_TAG_DISCOVER_CTRL) | \
   BIT (PXEBC_VENDOR_TAG_DISCOVER_MCAST) | \
   BIT (PXEBC_VENDOR_TAG_BOOT_SERVERS) | \
   BIT (PXEBC_VENDOR_TAG_BOOT_MENU) | \
   BIT (PXEBC_VENDOR_TAG_MENU_PROMPT))
```

#define GET_NEXT_BOOT_SVR_ENTRY (Ent)

Value:

```
(PXEBC_BOOT_SVR_ENTRY *) ((UINT8 *) Ent + sizeof (*(Ent)) + \
   ((Ent)->IpCnt - 1) * sizeof (EFI_IPv4_ADDRESS))
```

Referenced by PxeBcExtractDiscoverInfo().

#define GET_NEXT_DHCP_OPTION (Opt)

Value:

```
(EFI_DHCP4_PACKET_OPTION *) ((UINT8 *) (Opt) + \
   sizeof (EFI_DHCP4_PACKET_OPTION) + (Opt)->Length - 1)
```

#define GET_OPTION_BUFFER_LEN (Pkt) ((Pkt)->Length - sizeof (EFI_DHCP4_HEADER) - 4)

#define IS_DISABLE_BCAST_DISCOVER (x) (((x) & BIT (0)) == BIT (0))

Referenced by PxeBcExtractDiscoverInfo().

#define IS_DISABLE_MCAST_DISCOVER (x) (((x) & BIT (1)) == BIT (1))

Referenced by PxeBcExtractDiscoverInfo().

#define IS_DISABLE_PROMPT_MENU (x) (((x) & BIT (3)) == BIT (3))

Referenced by PxeBcDhcp4BootInfo(), and PxeBcSelectBootPrompt().

#define IS_ENABLE_USE_SERVER_LIST (x) (((x) & BIT (2)) == BIT (2))

Referenced by PxeBcExtractDiscoverInfo().

#define IS_PROXY_DHCP_OFFER (Offer) EFI_IP4_EQUAL (&(Offer)->Dhcp4.Header.YourAddr, &mZeroIp4Addr)

Referenced by PxeBcCacheDhcp4Offer(), PxeBcHandleDhcp4Offer(), and PxeBcSelectDhcp4Offer().

#define IS_VALID_BOOT_MENU (x)

Value:

```
((((x)[0]) & BIT (PXEBC_VENDOR_TAG_BOOT_MENU)) \
   == BIT (PXEBC_VENDOR_TAG_BOOT_MENU))
```

Referenced by PxeBcSelectBootMenu().

#define IS_VALID_BOOT_PROMPT (x)

Value:

```
((((x)[0]) & BIT (PXEBC_VENDOR_TAG_MENU_PROMPT)) \
   == BIT (PXEBC_VENDOR_TAG_MENU_PROMPT))
```

Referenced by PxeBcSelectBootPrompt().

#define IS_VALID_BOOT_SERVERS (x)

Value:

```
((((x)[0]) & BIT (PXEBC_VENDOR_TAG_BOOT_SERVERS)) \
   == BIT (PXEBC_VENDOR_TAG_BOOT_SERVERS))
```

Referenced by PxeBcDhcp4BootInfo(), and PxeBcExtractDiscoverInfo().

#define IS_VALID_BOOTITEM_VENDOR_OPTION (x)

Value:

```
(((UINT32) ((x)[PXEBC_VENDOR_TAG_BOOT_ITEM / 32]) & \
     BIT (PXEBC_VENDOR_TAG_BOOT_ITEM % 32)) \
```

```
    == BIT (PXEBC_VENDOR_TAG_BOOT_ITEM % 32))
```

#define IS_VALID_CREDENTIAL_VENDOR_OPTION (x)

Value:

```
(((UINT32) ((x)[0]) & BIT (PXEBC_VENDOR_TAG_CREDENTIAL_TYPES)) \
   == BIT (PXEBC_VENDOR_TAG_CREDENTIAL_TYPES))
```

#define IS_VALID_DISCOVER_VENDOR_OPTION (x) (((UINT32) ((x)[0]) & DISCOVER_VENDOR_OPTION_BIT_MAP) != 0)

Referenced by PxeBcExtractDiscoverInfo(), and PxeBcParseDhcp4Packet().

#define IS_VALID_MTFTP_VENDOR_OPTION (x)

Value:

```
(((UINT32) ((x)[0]) & MTFTP_VENDOR_OPTION_BIT_MAP) \
   == MTFTP_VENDOR_OPTION_BIT_MAP)
```

Referenced by PxeBcParseDhcp4Packet().

#define MTFTP_VENDOR_OPTION_BIT_MAP

Value:

```
(BIT (PXEBC_VENDOR_TAG_MTFTP_IP) | \
   BIT (PXEBC_VENDOR_TAG_MTFTP_CPORT) | \
   BIT (PXEBC_VENDOR_TAG_MTFTP_SPORT) | \
   BIT (PXEBC_VENDOR_TAG_MTFTP_TIMEOUT) | \
   BIT (PXEBC_VENDOR_TAG_MTFTP_DELAY))
```

#define PXEBC_BOOT_REQUEST_RETRIES 4

Referenced by PxeBcDhcp4Discover().

#define PXEBC_BOOT_REQUEST_TIMEOUT 1

Referenced by PxeBcDhcp4Discover().

#define PXEBC_BS_DISCOVER_PORT 4011

Referenced by PxeBcDhcp4Discover(), PxeBcDhcp6Discover(), and PxeBcRequestBootService().

#define PXEBC_BS_DOWNLOAD_PORT 69

Referenced by PxeBcMtftp4GetFileSize(), PxeBcMtftp4ReadDirectory(), PxeBcMtftp4ReadFile(), PxeBcMtftp4WriteFile(), PxeBcMtftp6GetFileSize(), PxeBcMtftp6ReadDirectory(), PxeBcMtftp6ReadFile(), and PxeBcMtftp6WriteFile().

#define PXEBC_DHCP4_C_PORT 68

#define PXEBC_DHCP4_MAGIC 0x63538263

Referenced by PxeBcSeedDhcp4Packet().

#define PXEBC_DHCP4_MSG_TYPE_REQUEST 3

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_OPCODE_REPLY 2

#define PXEBC_DHCP4_OPCODE_REQUEST 1

Referenced by PxeBcSeedDhcp4Packet().

#define PXEBC_DHCP4_OPTION_MAX_NUM 16

Referenced by PxeBcDhcp4Discover(), and PxeBcDhcp4Dora().

#define PXEBC_DHCP4_OPTION_MAX_SIZE 312

Referenced by PxeBcDhcp4Discover(), and PxeBcDhcp4Dora().

#define PXEBC_DHCP4_OVERLOAD_FILE 1

Referenced by PxeBcParseDhcp4Packet().

#define PXEBC_DHCP4_OVERLOAD_SERVER_NAME 2

Referenced by PxeBcParseDhcp4Packet().

#define PXEBC_DHCP4_PACKET_MAX_SIZE 1472

Referenced by EfiPxeBcStart(), PxeBcBuildDhcp4Options(), and PxeBcDhcp4CallBack().

#define PXEBC_DHCP4_S_PORT 67

Referenced by PxeBcDhcp4Discover().

#define PXEBC_DHCP4_TAG_BOOTFILE 67

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_BOOTFILE_LEN 13

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_BROADCAST 28

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_CLASS_ID 60

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_CLIENT_ID 61

#define PXEBC_DHCP4_TAG_DNS_SERVER 6

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_DOMAINNAME 15

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_DUMP 14

#define PXEBC_DHCP4_TAG_EMTU 22

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_EOP 255

Referenced by PxeBcDhcp4Discover(), PxeBcParseDhcp4Options(), PxeBcParseVendorOptions(), and PxeBcSeedDhcp4Packet().

#define PXEBC_DHCP4_TAG_EXTEND_PATH 18

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_HOSTNAME 12

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_INDEX_BOOTFILE 6

Referenced by PxeBcDhcp4BootInfo(), PxeBcHandleDhcp4Offer(), PxeBcParseDhcp4Packet(), PxeBcRetryBinlOffer(), PxeBcSelectBootPrompt(), and PxeBcSelectDhcp4Offer().

#define PXEBC_DHCP4_TAG_INDEX_BOOTFILE_LEN 0

Referenced by PxeBcDhcp4BootInfo().

#define PXEBC_DHCP4_TAG_INDEX_CLASS_ID 5

Referenced by PxeBcParseDhcp4Packet().

#define PXEBC_DHCP4_TAG_INDEX_MAX 7

Referenced by PxeBcParseDhcp4Packet().

#define PXEBC_DHCP4_TAG_INDEX_MSG_TYPE 3

Referenced by PxeBcParseDhcp4Packet().

#define PXEBC_DHCP4_TAG_INDEX_OVERLOAD 2

Referenced by PxeBcParseDhcp4Packet().

#define PXEBC_DHCP4_TAG_INDEX_SERVER_ID 4

Referenced by PxeBcDhcp4BootInfo(), and PxeBcRetryBinlOffer().

#define PXEBC_DHCP4_TAG_INDEX_VENDOR 1

Referenced by PxeBcParseDhcp4Packet().

#define PXEBC_DHCP4_TAG_LEASE 51

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_MAXMSG 57

Referenced by PxeBcBuildDhcp4Options(), and PxeBcDhcp4CallBack().

#define PXEBC_DHCP4_TAG_MSG_TYPE 53

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_NAME_SERVER 5

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_NETMASK 1

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_NIS_DOMAIN 40

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_NIS_SERVER 41

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_NTP_SERVER 42

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_OVERLOAD 52

#define PXEBC_DHCP4_TAG_PAD 0

Referenced by PxeBcParseDhcp4Options(), and PxeBcParseVendorOptions().

#define PXEBC_DHCP4_TAG_PARA_LIST 55

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_REQUEST_IP 50

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_ROOTPATH 17

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_ROUTER 3

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_SERVER_ID 54

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_T1 58

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_T2 59

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_TFTP 66

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_TIME_OFFSET 2

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_TIME_SERVER 4

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_TTL 23

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_DHCP4_TAG_VENDOR 43

Referenced by PxeBcBuildDhcp4Options(), and PxeBcDhcp4Discover().

#define PXEBC_PXE_DHCP4_TAG_ARCH 93

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_PXE_DHCP4_TAG_UNDI 94

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_PXE_DHCP4_TAG_UUID 97

Referenced by PxeBcBuildDhcp4Options().

#define PXEBC_VENDOR_TAG_BOOT_ITEM 71

Referenced by PxeBcDhcp4Discover(), and PxeBcParseVendorOptions().

#define PXEBC_VENDOR_TAG_BOOT_MENU 9

Referenced by PxeBcParseVendorOptions().

#define PXEBC_VENDOR_TAG_BOOT_SERVERS 8

Referenced by PxeBcParseVendorOptions().

#define PXEBC_VENDOR_TAG_CREDENTIAL_TYPES 12

Referenced by PxeBcParseVendorOptions().

#define PXEBC_VENDOR_TAG_DISCOVER_CTRL 6

Referenced by PxeBcParseVendorOptions().

#define PXEBC_VENDOR_TAG_DISCOVER_MCAST 7

Referenced by PxeBcParseVendorOptions().

#define PXEBC_VENDOR_TAG_MCAST_ALLOC 11

Referenced by PxeBcParseVendorOptions().

#define PXEBC_VENDOR_TAG_MENU_PROMPT 10

Referenced by PxeBcParseVendorOptions().

#define PXEBC_VENDOR_TAG_MTFTP_CPORT 2

Referenced by PxeBcParseVendorOptions().

#define PXEBC_VENDOR_TAG_MTFTP_DELAY 5

Referenced by PxeBcParseVendorOptions().

#define PXEBC_VENDOR_TAG_MTFTP_IP 1

Referenced by PxeBcParseVendorOptions().

#define PXEBC_VENDOR_TAG_MTFTP_SPORT 3

Referenced by PxeBcParseVendorOptions().

#define PXEBC_VENDOR_TAG_MTFTP_TIMEOUT 4

Referenced by PxeBcParseVendorOptions().

#define SET_VENDOR_OPTION_BIT_MAP (x, y) (*(x + ((y) / 32)) = (UINT32) ((UINT32) ((x)[(y) / 32]) | BIT ((y) % 32)))

Referenced by PxeBcParseVendorOptions().

Enumeration Type Documentation

enum PXEBC_OFFER_TYPE

Enumerator:

PxeOfferTypeDhcpOnly
PxeOfferTypeDhcpPxe10
PxeOfferTypeDhcpWfm11a
PxeOfferTypeDhcpBinl
PxeOfferTypeProxyPxe10
PxeOfferTypeProxyWfm11a
PxeOfferTypeProxyBinl
PxeOfferTypeBootp
PxeOfferTypeMax

Function Documentation

EFI_STATUS PxeBcDhcp4Discover (IN PXEBC_PRIVATE_DATA * *Private*,
IN UINT16 *Type*,
IN UINT16 * *Layer*,
IN BOOLEAN *UseBis*,

```
                    IN EFI_IP_ADDRESS *                 DestIp,
                    IN UINT16                           IpCount,
                    IN EFI_PXE_BASE_CODE_SRVLIST *      SrvList
                  )
```

Build and send out the request packet for the bootfile, and parse the reply.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Type*	PxeBc option boot item type.
[in]	*Layer*	Pointer to option boot item layer.
[in]	*UseBis*	Use BIS or not.
[in]	*DestIp*	Pointer to the server address.
[in]	*IpCount*	The total count of the server address.
[in]	*SrvList*	Pointer to EFI_PXE_BASE_CODE_SRVLIST.

Return values:

EFI_SUCCESS	Successfully discovered boot file.
EFI_OUT_OF_RESOURCES	Failed to allocate resource.
EFI_NOT_FOUND	Can't get the PXE reply packet.
Others	Failed to discover boot file.

References PXEBC_OPTION_BOOT_ITEM::Layer, PXEBC_BOOT_REQUEST_RETRIES, PXEBC_BOOT_REQUEST_TIMEOUT, PXEBC_BS_DISCOVER_PORT, PXEBC_DHCP4_OPTION_MAX_NUM, PXEBC_DHCP4_OPTION_MAX_SIZE, PXEBC_DHCP4_S_PORT, PXEBC_DHCP4_TAG_EOP, PXEBC_DHCP4_TAG_VENDOR, PXEBC_VENDOR_TAG_BOOT_ITEM, PxeBcBuildDhcp4Options(), PxeBcCacheDhcp4Packet(), and PXEBC_OPTION_BOOT_ITEM::Type.

Referenced by PxeBcDiscoverBootServer(), and PxeBcRetryBinlOffer().

```
EFI_STATUS PxeBcDhcp4Dora ( IN PXEBC_PRIVATE_DATA *   Private,
                            IN EFI_DHCP4_PROTOCOL *   Dhcp4
                          )
```

Start the D.O.R.A DHCPv4 process to acquire the IPv4 address and other PXE boot information.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Dhcp4*	Pointer to the EFI_DHCP4_PROTOCOL

Return values:

EFI_SUCCESS	The D.O.R.A process successfully finished.
Others	Failed to finish the D.O.R.A process.

References mPxeDhcpTimeout, PXEBC_DHCP4_OPTION_MAX_NUM, PXEBC_DHCP4_OPTION_MAX_SIZE, PXEBC_DHCP_RETRIES, PxeBcBuildDhcp4Options(), PxeBcDhcp4CallBack(), PxeBcFlushStationIp(), PxeBcHandleDhcp4Offer(), and PxeBcShowIp4Addr().

Referenced by EfiPxeBcDhcp().

```
EFI_STATUS PxeBcParseDhcp4Packet ( IN PXEBC_DHCP4_PACKET_CACHE *   Cache4   )
```

Parse the cached DHCPv4 packet, including all the options.

Parameters:

[in] *Cache4* Pointer to cached DHCPv4 packet.

Return values:

EFI_SUCCESS Parsed the DHCPv4 packet successfully.
EFI_DEVICE_ERROR Failed to parse and invalid packet.

Parse the cached DHCPv4 packet, including all the options.

Parameters:

[in] *Cache4* Pointer to cached DHCPv4 packet.

Return values:

EFI_SUCCESS Parsed the DHCPv4 packet successfully.
EFI_DEVICE_ERROR Failed to parse and invalid packet.

References DEFAULT_CLASS_ID_DATA, GET_OPTION_BUFFER_LEN, IS_VALID_DISCOVER_VENDOR_OPTION, IS_VALID_MTFTP_VENDOR_OPTION, mInterestedDhcp4Tags, PXEBC_DHCP4_OVERLOAD_FILE, PXEBC_DHCP4_OVERLOAD_SERVER_NAME, PXEBC_DHCP4_TAG_INDEX_BOOTFILE, PXEBC_DHCP4_TAG_INDEX_CLASS_ID, PXEBC_DHCP4_TAG_INDEX_MAX, PXEBC_DHCP4_TAG_INDEX_MSG_TYPE, PXEBC_DHCP4_TAG_INDEX_OVERLOAD, PXEBC_DHCP4_TAG_INDEX_VENDOR, PxeBcParseDhcp4Options(), PxeBcParseVendorOptions(), PxeOfferTypeBootp, PxeOfferTypeDhcpBinl, PxeOfferTypeDhcpOnly, PxeOfferTypeDhcpPxe10, PxeOfferTypeProxyBinl, and PxeOfferTypeProxyPxe10.

Referenced by EfiPxeBcDiscover(), PxeBcCacheDhcp4Offer(), PxeBcCopyDhcp4Ack(), PxeBcCopyProxyOffer(), and PxeBcRetryBinlOffer().

```
VOID PxeBcSeedDhcp4Packet ( OUT EFI_DHCP4_PACKET *   Seed,
                            IN EFI_UDP4_PROTOCOL *   Udp4
                          )
```

Create a template DHCPv4 packet as a seed.

Parameters:

[out] *Seed* Pointer to the seed packet.
[in] *Udp4* Pointer to EFI_UDP4_PROTOCOL.

References PXEBC_DHCP4_MAGIC, PXEBC_DHCP4_OPCODE_REQUEST, and PXEBC_DHCP4_TAG_EOP.

Referenced by EfiPxeBcStart().

```
EFI_STATUS PxeBcSetIp4Policy ( IN PXEBC_PRIVATE_DATA * Private )
```

Switch the Ip4 policy to static.

Parameters:

`[in]` *Private* The pointer to PXEBC_PRIVATE_DATA.

Return values:

EFI_SUCCESS The policy is already configured to static.
Others Other error as indicated..

References Policy.

Referenced by EfiPxeBcStart().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/UefiPxeBcDxe/PxeBcDhcp6.c File Reference

Functions

EFI_DHCP6_PACKET_OPTION * PxeBcParseDhcp6Options (IN UINT8 *Buffer, IN UINT32 Length, IN UINT16 OptTag)
UINT32 PxeBcBuildDhcp6Options (IN PXEBC_PRIVATE_DATA *Private, OUT EFI_DHCP6_PACKET_OPTION **OptList, IN UINT8 *Buffer)
VOID PxeBcCacheDhcp6Packet (IN EFI_DHCP6_PACKET *Dst, IN EFI_DHCP6_PACKET *Src)
VOID PxeBcFreeBootFileOption (IN LIST_ENTRY *Head)
EFI_STATUS PxeBcExtractBootFileUrl (OUT UINT8 **FileName, IN OUT EFI_IPv6_ADDRESS *SrvAddr, IN CHAR8 *BootFile, IN UINT16 Length)
EFI_STATUS PxeBcExtractBootFileParam (IN CHAR8 *BootFilePara, OUT UINT16 *BootFileSize)
EFI_STATUS PxeBcParseDhcp6Packet (IN PXEBC_DHCP6_PACKET_CACHE *Cache6)
VOID PxeBcCopyDhcp6Ack (IN PXEBC_PRIVATE_DATA *Private, IN EFI_DHCP6_PACKET *Ack, IN BOOLEAN Verified)
VOID PxeBcCopyDhcp6Proxy (IN PXEBC_PRIVATE_DATA *Private, IN UINT32 OfferIndex)
UINT8 * PxeBcDhcp6SeekOption (IN UINT8 *Buf, IN UINT32 SeekLen, IN UINT16 OptType)
EFI_STATUS PxeBcRequestBootService (IN PXEBC_PRIVATE_DATA *Private, IN UINT32 Index)
EFI_STATUS PxeBcRetryDhcp6Binl (IN PXEBC_PRIVATE_DATA *Private, IN UINT32 Index)
VOID PxeBcCacheDhcp6Offer (IN PXEBC_PRIVATE_DATA *Private, IN EFI_DHCP6_PACKET *RcvdOffer)
VOID PxeBcSelectDhcp6Offer (IN PXEBC_PRIVATE_DATA *Private)
EFI_STATUS PxeBcHandleDhcp6Offer (IN PXEBC_PRIVATE_DATA *Private)
VOID PxeBcUnregisterIp6Address (IN PXEBC_PRIVATE_DATA *Private)
EFI_STATUS PxeBcCheckRouteTable (IN PXEBC_PRIVATE_DATA *Private, IN UINTN TimeOutInSecond, OUT EFI_IPv6_ADDRESS *GatewayAddr)
EFI_STATUS PxeBcRegisterIp6Address (IN PXEBC_PRIVATE_DATA *Private, IN EFI_IPv6_ADDRESS *Address)
EFI_STATUS PxeBcSetIp6Policy (IN PXEBC_PRIVATE_DATA *Private)
EFI_STATUS PxeBcSetIp6Address (IN PXEBC_PRIVATE_DATA *Private)
EFI_STATUS EFIAPI PxeBcDhcp6CallBack (IN EFI_DHCP6_PROTOCOL *This, IN VOID *Context, IN EFI_DHCP6_STATE CurrentState, IN EFI_DHCP6_EVENT Dhcp6Event, IN EFI_DHCP6_PACKET *Packet, OUT EFI_DHCP6_PACKET **NewPacket)
EFI_STATUS PxeBcDhcp6Discover (IN PXEBC_PRIVATE_DATA *Private, IN UINT16 Type, IN UINT16 *Layer, IN BOOLEAN UseBis, IN EFI_IP_ADDRESS *DestIp)
EFI_STATUS PxeBcDhcp6Sarr (IN PXEBC_PRIVATE_DATA *Private, IN EFI_DHCP6_PROTOCOL *Dhcp6)

Variables

EFI_IPv6_ADDRESS mAllDhcpRelayAndServersAddress = {{0xFF, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2}}

Detailed Description

Functions implementation related with DHCPv6 for UefiPxeBc Driver.

(C) Copyright 2014 Hewlett-Packard Development Company, L.P.
Copyright (c) 2009 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
UINT32 PxeBcBuildDhcp6Options ( IN PXEBC_PRIVATE_DATA *              Private,
                                OUT EFI_DHCP6_PACKET_OPTION **       OptList,
                                IN UINT8 *                           Buffer
                              )
```

Build the options buffer for the DHCPv6 request packet.

Parameters:

[in]	*Private*	The pointer to PxeBc private data.
[out]	*OptList*	The pointer to the option pointer array.
[in]	*Buffer*	The pointer to the buffer to contain the option list.

Returns:

Index The count of the built-in options.

References PXEBC_DHCP6_OPTION_ENTRY::Arch, PXEBC_CLASS_ID::ArchitectureType, PXEBC_DHCP6_OPTION_VENDOR_CLASS::ClassId, PXEBC_DHCP6_OPTION_VENDOR_CLASS::ClassLen, DEFAULT_CLASS_ID_DATA, DEFAULT_UNDI_MAJOR, DEFAULT_UNDI_MINOR, DEFAULT_UNDI_TYPE, GET_NEXT_DHCP6_OPTION, PXEBC_CLASS_ID::InterfaceName, PXEBC_DHCP6_OPTION_UNDI::MajorVer, PXEBC_DHCP6_OPTION_UNDI::MinorVer, PXEBC_DHCP6_OPTION_ORO::OpCode, PXEBC_DHCP6_OPTION_ENTRY::Oro, PXEBC_DHCP6_ENTERPRISE_NUM, PXEBC_DHCP6_OPT_ARCH, PXEBC_DHCP6_OPT_BOOT_FILE_PARAM, PXEBC_DHCP6_OPT_BOOT_FILE_URL, PXEBC_DHCP6_OPT_ORO, PXEBC_DHCP6_OPT_UNDI, PXEBC_DHCP6_OPT_VENDOR_CLASS, PxeBcUintnToAscDecWithFormat(), PXEBC_DHCP6_OPTION_ARCH::Type, PXEBC_DHCP6_OPTION_UNDI::Type, PXEBC_DHCP6_OPTION_ENTRY::Undi, PXEBC_CLASS_ID::UndiMajor, PXEBC_CLASS_ID::UndiMinor, PXEBC_DHCP6_OPTION_VENDOR_CLASS::Vendor, and PXEBC_DHCP6_OPTION_ENTRY::VendorClass.

Referenced by PxeBcDhcp6Sarr().

```
VOID PxeBcCacheDhcp6Offer ( IN PXEBC_PRIVATE_DATA *   Private,
                            IN EFI_DHCP6_PACKET *     RcvdOffer
                          )
```

Cache all the received DHCPv6 offers, and set OfferIndex and OfferCount.

Parameters:

[in]	*Private*	The pointer to PXEBC_PRIVATE_DATA.
[in]	*RcvdOffer*	The pointer to the received offer packet.

References IS_PROXY_OFFER, PXEBC_DHCP6_PACKET::Offer, PXEBC_DHCP6_PACKET_CACHE::OfferType, PXEBC_DHCP6_PACKET_CACHE::Packet, PXEBC_OFFER_MAX_NUM, PxeBcCacheDhcp6Packet(), PxeBcParseDhcp6Packet(), PxeOfferTypeMax, and PxeOfferTypeProxyBinl.

Referenced by PxeBcDhcp6CallBack().

```
VOID PxeBcCacheDhcp6Packet ( IN EFI_DHCP6_PACKET *   Dst,
                             IN EFI_DHCP6_PACKET *   Src
                           )
```

Cache the DHCPv6 packet.

Parameters:

[in]	*Dst*	The pointer to the cache buffer for DHCPv6 packet.
[in]	*Src*	The pointer to the DHCPv6 packet to be cached.

Referenced by PxeBcCacheDhcp6Offer(), PxeBcCopyDhcp6Ack(), and PxeBcCopyDhcp6Proxy().

```
EFI_STATUS PxeBcCheckRouteTable ( IN PXEBC_PRIVATE_DATA *   Private,
                                  IN UINTN                  TimeOutInSecond,
                                  OUT EFI_IPv6_ADDRESS *    GatewayAddr
                                )
```

Check whether IP driver could route the message which will be sent to ServerIp address.

This function will check the IP6 route table every 1 seconds until specified timeout is expired, if a valid route is found in IP6 route table, the address will be filed in GatewayAddr and return.

Parameters:

[in]	*Private*	The pointer to PXEBC_PRIVATE_DATA.
[in]	*TimeOutInSecond*	Timeout value in seconds.
[out]	*GatewayAddr*	Pointer to store the gateway IP address.

Return values:

EFI_SUCCESS	Found a valid gateway address successfully.

EFI_TIMEOUT The operation is time out.
Other Unexpect error happened.

Referenced by PxeBcRegisterIp6Address().

```
VOID PxeBcCopyDhcp6Ack ( IN PXEBC_PRIVATE_DATA *   Private,
                         IN EFI_DHCP6_PACKET *     Ack,
                         IN BOOLEAN                Verified
                       )
```

Cache the DHCPv6 ack packet, and parse it on demand.

Parameters:

[in] *Private* The pointer to PxeBc private data.
[in] *Ack* The pointer to the DHCPv6 ack packet.
[in] *Verified* If TRUE, parse the ACK packet and store info into mode data.

References PxeBcCacheDhcp6Packet(), and PxeBcParseDhcp6Packet().

Referenced by PxeBcDhcp6CallBack(), and PxeBcHandleDhcp6Offer().

```
VOID PxeBcCopyDhcp6Proxy ( IN PXEBC_PRIVATE_DATA *   Private,
                           IN UINT32                 OfferIndex
                         )
```

Cache the DHCPv6 proxy offer packet according to the received order.

Parameters:

[in] *Private* The pointer to PxeBc private data.
[in] *OfferIndex* The received order of offer packets.

References PXEBC_OFFER_MAX_NUM, PxeBcCacheDhcp6Packet(), and PxeBcParseDhcp6Packet().

Referenced by PxeBcHandleDhcp6Offer().

```
EFI_STATUS EFIAPI PxeBcDhcp6CallBack ( IN EFI_DHCP6_PROTOCOL *   This,
                                       IN VOID *                 Context,
                                       IN EFI_DHCP6_STATE        CurrentState,
                                       IN EFI_DHCP6_EVENT        Dhcp6Event,
                                       IN EFI_DHCP6_PACKET *     Packet,
                                       OUT EFI_DHCP6_PACKET **   NewPacket
                                     )
```

EFI_DHCP6_CALLBACK is provided by the consumer of the EFI DHCPv6 Protocol driver to intercept events that occurred in the configuration process.

Parameters:

[in] *This* The pointer to the EFI DHCPv6 Protocol.
[in] *Context* The pointer to the context set by EFI_DHCP6_PROTOCOL.Configure().
[in] *CurrentState* The current operational state of the EFI DHCPv Protocol driver.
[in] *Dhcp6Event*

		The event that occurs in the current state, which usually means a state transition.
[in]	*Packet*	The DHCPv6 packet that is going to be sent or was already received.
[out]	*NewPacket*	The packet that is used to replace the Packet above.

Return values:

EFI_SUCCESS	Told the EFI DHCPv6 Protocol driver to continue the DHCP process.
EFI_NOT_READY	Only used in the Dhcp6Selecting state. The EFI DHCPv6 Protocol driver will continue to wait for more packets.
EFI_ABORTED	Told the EFI DHCPv6 Protocol driver to abort the current process.

References CalcElapsedTime(), PXEBC_DHCP_PACKET_CACHE::Dhcp6, _PXEBC_PRIVATE_DATA::Dhcp6Request, _PXEBC_PRIVATE_DATA::Function, PXEBC_DHCP6_PACKET::Offer, _PXEBC_PRIVATE_DATA::OfferBuffer, _PXEBC_PRIVATE_DATA::OfferNum, PXEBC_DHCP6_PACKET_CACHE::Packet, _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_OFFER_MAX_NUM, PxeBcCacheDhcp6Offer(), _PXEBC_PRIVATE_DATA::PxeBcCallback, PxeBcCopyDhcp6Ack(), PxeBcSelectDhcp6Offer(), _PXEBC_PRIVATE_DATA::SelectIndex, and _PXEBC_PRIVATE_DATA::SolicitTimes.

Referenced by PxeBcDhcp6Sarr().

```
EFI_STATUS PxeBcDhcp6Discover ( IN PXEBC_PRIVATE_DATA *   Private,
                                IN UINT16                 Type,
                                IN UINT16 *               Layer,
                                IN BOOLEAN                UseBis,
                                IN EFI_IP_ADDRESS *       DestIp
                              )
```

Build and send out the request packet for the bootfile, and parse the reply.

Parameters:

[in]	*Private*	The pointer to PxeBc private data.
[in]	*Type*	PxeBc option boot item type.
[in]	*Layer*	The pointer to option boot item layer.
[in]	*UseBis*	Use BIS or not.
[in]	*DestIp*	The pointer to the server address.

Return values:

EFI_SUCCESS	Successfully discovered the boot file.
EFI_OUT_OF_RESOURCES	Failed to allocate resources.
EFI_NOT_FOUND	Can't get the PXE reply packet.
Others	Failed to discover the boot file.

References PXEBC_BS_DISCOVER_PORT.

Referenced by PxeBcDiscoverBootServer().

```
EFI_STATUS PxeBcDhcp6Sarr ( IN PXEBC_PRIVATE_DATA *   Private,
                            IN EFI_DHCP6_PROTOCOL *   Dhcp6
                          )
```

Start the DHCPv6 S.A.R.R. process to acquire the IPv6 address and other PXE boot information.

Parameters:

[in] *Private* The pointer to PxeBc private data.
[in] *Dhcp6* The pointer to the EFI_DHCP6_PROTOCOL

Return values:

EFI_SUCCESS The S.A.R.R. process successfully finished.
Others Failed to finish the S.A.R.R. process.

References PXEBC_DAD_ADDITIONAL_DELAY, PXEBC_DHCP6_OPTION_MAX_NUM, PXEBC_DHCP6_OPTION_MAX_SIZE, PxeBcBuildDhcp6Options(), PxeBcDhcp6CallBack(), and PxeBcHandleDhcp6Offer().

Referenced by EfiPxeBcDhcp().

```
UINT8* PxeBcDhcp6SeekOption ( IN UINT8 *   Buf,
                              IN UINT32    SeekLen,
                              IN UINT16    OptType
                            )
```

Seek the address of the first byte of the option header.

Parameters:

[in] *Buf* The pointer to the buffer.
[in] *SeekLen* The length to seek.
[in] *OptType* The option type.

Return values:

NULL If it failed to seek the option.
others The position to the option.

Referenced by PxeBcRequestBootService().

```
EFI_STATUS PxeBcExtractBootFileParam ( IN CHAR8 *      BootFilePara,
                                       OUT UINT16 *    BootFileSize
                                     )
```

Parse the Boot File Parameter option.

Parameters:

[in] *BootFilePara* The pointer to boot file parameter option data.
[out] *BootFileSize* The pointer to the parsed boot file size.

Return values:

EFI_SUCCESS Successfully obtained the boot file size from parameter option.
EFI_NOT_FOUND Failed to extract the boot file size from parameter option.

References PXEBC_DHCP6_MAX_BOOT_FILE_SIZE, and PxeBcUniHexToUint8().

Referenced by PxeBcDhcp6BootInfo().

```
EFI_STATUS PxeBcExtractBootFileUrl ( OUT UINT8 **                FileName,
                                     IN OUT EFI_IPv6_ADDRESS *   SrvAddr,
                                     IN CHAR8 *                  BootFile,
                                     IN UINT16                   Length
                                   )
```

Parse the Boot File URL option.

Parameters:

[out]	*FileName*	The pointer to the boot file name.
[in,out]	*SrvAddr*	The pointer to the boot server address.
[in]	*BootFile*	The pointer to the boot file URL option data.
[in]	*Length*	The length of the boot file URL option data.

Return values:

EFI_ABORTED	User cancel operation.
EFI_SUCCESS	Selected the boot menu successfully.
EFI_NOT_READY	Read the input key from the keybroad has not finish.

References PXEBC_ADDR_END_DELIMITER, PXEBC_ADDR_START_DELIMITER, PXEBC_DHCP6_BOOT_FILE_URL_PREFIX, and PXEBC_TFTP_URL_SEPARATOR.

Referenced by PxeBcDhcp6BootInfo(), and PxeBcRetryDhcp6Binl().

```
VOID PxeBcFreeBootFileOption ( IN LIST_ENTRY *   Head   )
```

Free all the nodes in the list for boot file.

Parameters:

[in]	*Head*	The pointer to the head of list.

```
EFI_STATUS PxeBcHandleDhcp6Offer   (  IN PXEBC_PRIVATE_DATA *   Private   )
```

Handle the DHCPv6 offer packet.

Parameters:

[in]	*Private*	The pointer to PXEBC_PRIVATE_DATA.

Return values:

EFI_SUCCESS	Handled the DHCPv6 offer packet successfully.
EFI_NO_RESPONSE	No response to the following request packet.

References IS_PROXY_OFFER, PXEBC_DHCP6_PACKET_CACHE::OfferType, PXEBC_DHCP6_PACKET_CACHE::OptList, PXEBC_DHCP6_IDX_BOOT_FILE_URL, PXEBC_OFFER_MAX_NUM, PxeBcCopyDhcp6Ack(), PxeBcCopyDhcp6Proxy(), PxeBcRetryDhcp6Binl(), PxeOfferTypeDhcpBinl, PxeOfferTypeDhcpOnly, and PxeOfferTypeProxyBinl.

Referenced by PxeBcDhcp6Sarr().

```
EFI_DHCP6_PACKET_OPTION* PxeBcParseDhcp6Options ( IN UINT8 *   Buffer,
```

IN UINT32 *Length*,
IN UINT16 *OptTag*
)

Parse out a DHCPv6 option by OptTag, and find the position in buffer.

Parameters:

[in] *Buffer* The pointer to the option buffer.
[in] *Length* Length of the option buffer.
[in] *OptTag* The required option tag.

Return values:

NULL Failed to parse the required option.
Others The postion of the required option in buffer.

Referenced by PxeBcParseDhcp6Packet().

EFI_STATUS PxeBcParseDhcp6Packet (IN PXEBC_DHCP6_PACKET_CACHE * *Cache6*)

Parse the cached DHCPv6 packet, including all the options.

Parameters:

[in] *Cache6* The pointer to a cached DHCPv6 packet.

Return values:

EFI_SUCCESS Parsed the DHCPv6 packet successfully.
EFI_DEVICE_ERROR Failed to parse and invalid the packet.

References DEFAULT_CLASS_ID_DATA, GET_DHCP6_OPTION_SIZE, PXEBC_DHCP6_ENTERPRISE_NUM, PXEBC_DHCP6_IDX_BOOT_FILE_PARAM, PXEBC_DHCP6_IDX_BOOT_FILE_URL, PXEBC_DHCP6_IDX_IA_NA, PXEBC_DHCP6_IDX_VENDOR_CLASS, PXEBC_DHCP6_OPT_BOOT_FILE_PARAM, PXEBC_DHCP6_OPT_BOOT_FILE_URL, PXEBC_DHCP6_OPT_IA_NA, PXEBC_DHCP6_OPT_STATUS_CODE, PXEBC_DHCP6_OPT_VENDOR_CLASS, PxeBcParseDhcp6Options(), PxeOfferTypeDhcpBinl, PxeOfferTypeDhcpOnly, and PxeOfferTypeProxyBinl.

Referenced by EfiPxeBcDiscover(), PxeBcCacheDhcp6Offer(), PxeBcCopyDhcp6Ack(), PxeBcCopyDhcp6Proxy(), and PxeBcRetryDhcp6Binl().

EFI_STATUS PxeBcRegisterIp6Address (IN PXEBC_PRIVATE_DATA * *Private*,
IN EFI_IPv6_ADDRESS * *Address*
)

Register the ready station address and gateway by Ip6Config protocol.

Parameters:

[in] *Private* The pointer to PXEBC_PRIVATE_DATA.
[in] *Address* The pointer to the ready address.

Return values:

EFI_SUCCESS Registered the address succesfully.
Others Failed to register the address.

References Policy, PXEBC_DAD_ADDITIONAL_DELAY, PXEBC_IP6_POLICY_MAX, PXEBC_IP6_ROUTE_TABLE_TIMEOUT, PxeBcCheckRouteTable(), and PxeBcCommonNotify().

Referenced by EfiPxeBcSetStationIP(), and PxeBcSetIp6Address().

```
EFI_STATUS PxeBcRequestBootService ( IN PXEBC_PRIVATE_DATA *  Private,
                                     IN UINT32                Index
                                   )
```

Build and send out the request packet for the bootfile, and parse the reply.

Parameters:

[in]	*Private*	The pointer to PxeBc private data.
[in]	*Index*	PxeBc option boot item type.

Return values:

EFI_SUCCESS	Successfully discovered the boot file.
EFI_OUT_OF_RESOURCES	Failed to allocate resources.
EFI_NOT_FOUND	Can't get the PXE reply packet.
Others	Failed to discover the boot file.

References CalcElapsedTime(), PXEBC_BS_DISCOVER_PORT, PXEBC_DHCP6_OPT_ELAPSED_TIME, PXEBC_DHCP6_OPT_SERVER_ID, and PxeBcDhcp6SeekOption().

Referenced by PxeBcRetryDhcp6Binl().

```
EFI_STATUS PxeBcRetryDhcp6Binl ( IN PXEBC_PRIVATE_DATA *  Private,
                                 IN UINT32                Index
                               )
```

Retry to request bootfile name by the BINL offer.

Parameters:

[in]	*Private*	The pointer to PxeBc private data.
[in]	*Index*	The received order of offer packets.

Return values:

EFI_SUCCESS	Successfully retried a request for the bootfile name.
EFI_DEVICE_ERROR	Failed to retry the bootfile name.

References mAllDhcpRelayAndServersAddress, PXEBC_DHCP6_PACKET::Offer, PXEBC_DHCP6_PACKET_CACHE::OfferType, PXEBC_DHCP6_PACKET_CACHE::OptList, PXEBC_DHCP6_PACKET_CACHE::Packet, PXEBC_DHCP6_IDX_BOOT_FILE_URL, PXEBC_OFFER_MAX_NUM, PxeBcExtractBootFileUrl(), PxeBcParseDhcp6Packet(), PxeBcRequestBootService(), PxeOfferTypeDhcpBinl, PxeOfferTypeProxyBinl, PxeOfferTypeProxyPxe10, and PxeOfferTypeProxyWfm11a.

Referenced by PxeBcHandleDhcp6Offer().

VOID PxeBcSelectDhcp6Offer (IN PXEBC_PRIVATE_DATA * *Private*)

Select an DHCPv6 offer, and record SelectIndex and SelectProxyType.

Parameters:

[in] *Private* The pointer to PXEBC_PRIVATE_DATA.

References IS_PROXY_OFFER, PXEBC_DHCP6_IDX_BOOT_FILE_URL, PxeOfferTypeDhcpBinl, PxeOfferTypeDhcpOnly, PxeOfferTypeDhcpPxe10, PxeOfferTypeDhcpWfm11a, PxeOfferTypeProxyBinl, PxeOfferTypeProxyPxe10, and PxeOfferTypeProxyWfm11a.

Referenced by PxeBcDhcp6CallBack().

EFI_STATUS PxeBcSetIp6Address (IN PXEBC_PRIVATE_DATA * *Private*)

This function will register the station IP address and flush IP instance to start using the new IP address.

Parameters:

[in] *Private* The pointer to PXEBC_PRIVATE_DATA.

Return values:

EFI_SUCCESS The new IP address has been configured successfully.

Others Failed to configure the address.

References PxeBcFlushStationIp(), PxeBcRegisterIp6Address(), PxeBcShowIp6Addr(), and PxeBcUnregisterIp6Address().

Referenced by PxeBcDhcp6BootInfo().

EFI_STATUS PxeBcSetIp6Policy (IN PXEBC_PRIVATE_DATA * *Private*)

Set the IP6 policy to Automatic.

Parameters:

[in] *Private* The pointer to PXEBC_PRIVATE_DATA.

Return values:

EFI_SUCCESS Switch the IP policy succesfully.

Others Unexpect error happened.

References Policy, and PXEBC_IP6_POLICY_MAX.

Referenced by EfiPxeBcStart().

VOID PxeBcUnregisterIp6Address (IN PXEBC_PRIVATE_DATA * *Private*)

Unregister the address by Ip6Config protocol.

Parameters:

[in] *Private* The pointer to PXEBC_PRIVATE_DATA.

References PXEBC_IP6_POLICY_MAX.

Referenced by EfiPxeBcStop(), and PxeBcSetIp6Address().

Variable Documentation

EFI_IPv6_ADDRESS mAllDhcpRelayAndServersAddress = {{0xFF, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2}}

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/UefiPxeBcDxe/PxeBcDhcp6.h File Reference

Data Structures

struct PXEBC_DHCP6_OPTION_ORO
struct PXEBC_DHCP6_OPTION_UNDI
struct PXEBC_DHCP6_OPTION_ARCH
struct PXEBC_CLASS_ID
struct PXEBC_DHCP6_OPTION_VEND
union PXEBC_DHCP6_OPTION_ENTR
struct PXEBC_DHCP6_OPTION_NODE
union PXEBC_DHCP6_PACKET
struct PXEBC_DHCP6_PACKET_CACH

Defines

"memItemRight" valign="bottom">PXEBC_DHCP6_OPTION_MAX_NUM 16
#define PXEBC_DHCP6_OPTION_MAX_
#define PXEBC_DHCP6_PACKET_MAX_
#define PXEBC_IP6_POLICY_MAX 0xff
#define PXEBC_IP6_ROUTE_TABLE_TI
#define PXEBC_DHCP6_S_PORT 547
#define PXEBC_DHCP6_C_PORT 546
#define PXEBC_DHCP6_OPT_CLIENT_I
#define PXEBC_DHCP6_OPT_SERVER_I
#define PXEBC_DHCP6_OPT_IA_NA 3
#define PXEBC_DHCP6_OPT_IA_TA 4
#define PXEBC_DHCP6_OPT_IAADDR
#define PXEBC_DHCP6_OPT_ORO 6
#define PXEBC_DHCP6_OPT_PREFEREN
#define PXEBC_DHCP6_OPT_ELAPSED_
#define PXEBC_DHCP6_OPT_REPLAY_
#define PXEBC_DHCP6_OPT_AUTH 11
#define PXEBC_DHCP6_OPT_UNICAST
#define PXEBC_DHCP6_OPT_STATUS_C
#define PXEBC_DHCP6_OPT_RAPID_CO
#define PXEBC_DHCP6_OPT_USER_CLA
#define PXEBC_DHCP6_OPT_VENDOR_
#define PXEBC_DHCP6_OPT_VENDOR_
#define PXEBC_DHCP6_OPT_INTERFAC
#define PXEBC_DHCP6_OPT_RECONFIG
#define PXEBC_DHCP6_OPT_RECONFIG
#define PXEBC_DHCP6_OPT_BOOT_FIL
#define PXEBC_DHCP6_OPT_BOOT_FIL
#define PXEBC_DHCP6_OPT_ARCH 61
#define PXEBC_DHCP6_OPT_UNDI 62
#define PXEBC_DHCP6_ENTERPRISE_N
#define PXEBC_DHCP6_MAX_BOOT_FI
#define PXEBC_DHCP6_IDX_IA_NA 0
#define PXEBC_DHCP6_IDX_BOOT_FIL

```
#define PXEBC_DHCP6_IDX_BOOT_FILE_PAR
#define PXEBC_DHCP6_IDX_VENDOR_CLASS
#define PXEBC_DHCP6_IDX_MAX   4
#define PXEBC_DHCP6_BOOT_FILE_URL_PRE
#define PXEBC_TFTP_URL_SEPARATOR   '/'
#define PXEBC_ADDR_START_DELIMITER   '['
#define PXEBC_ADDR_END_DELIMITER   ']'
#define GET_NEXT_DHCP6_OPTION(Opt)
#define GET_DHCP6_OPTION_SIZE(Pkt)   ((Pkt)-
        sizeof (EFI_DHCP6_HEADER))
#define IS_PROXY_OFFER(Type)
```

Functions

```
VOID PxeBcFreeBootFileOption (IN LIST_ENTR
EFI_STATUS PxeBcExtractBootFileUrl (OUT UINT8 **I
        OUT EFI_IPv6_ADDRESS *SrvAddr, IN C
        *BootFile, IN UINT16 Length)
EFI_STATUS PxeBcExtractBootFileParam (IN CHAR8 *
        OUT UINT16 *BootFileSize)
EFI_STATUS PxeBcParseDhcp6Packet (IN
        PXEBC_DHCP6_PACKET_CACHE *Cach
EFI_STATUS PxeBcRegisterIp6Address (IN
        PXEBC_PRIVATE_DATA *Private, IN
        EFI_IPv6_ADDRESS *Address)
VOID PxeBcUnregisterIp6Address (IN
        PXEBC_PRIVATE_DATA *Private)
EFI_STATUS PxeBcDhcp6Discover (IN PXEBC_PRIVA
        *Private, IN UINT16 Type, IN UINT16 *La
        BOOLEAN UseBis, IN EFI_IP_ADDRESS
EFI_STATUS PxeBcSetIp6Policy (IN PXEBC_PRIVATE
        *Private)
EFI_STATUS PxeBcSetIp6Address (IN PXEBC_PRIVAT
        *Private)
EFI_STATUS PxeBcDhcp6Sarr (IN PXEBC_PRIVATE_I
        *Private, IN EFI_DHCP6_PROTOCOL *D
```

Detailed Description

Functions declaration related with DHCPv6 for UefiPxeBc Driver.

Copyright (c) 2009 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"paramtype">Pkt) ((Pkt)->Length - sizeof (EFI_DHCP6_HEADER))

Referenced by PxeBcParseDhcp6Packet().

#define GET_NEXT_DHCP6_OPTION (Opt)

Value:

```
(EFI_DHCP6_PACKET_OPTION *) ((UINT8 *) (Opt) + \
  sizeof (EFI_DHCP6_PACKET_OPTION) + (NTOHS ((Opt)->OpLen)) - 1)
```

Referenced by PxeBcBuildDhcp6Options().

#define IS_PROXY_OFFER (Type)

Value:

```
((Type) == PxeOfferTypeProxyBinl || \
   (Type) == PxeOfferTypeProxyPxe10 || \
   (Type) == PxeOfferTypeProxyWfm11a)
```

Referenced by PxeBcCacheDhcp6Offer(), PxeBcHandleDhcp6Offer(), and PxeBcSelectDhcp6Offer().

#define PXEBC_ADDR_END_DELIMITER ']'

Referenced by PxeBcExtractBootFileUrl().

#define PXEBC_ADDR_START_DELIMITER '['

Referenced by PxeBcExtractBootFileUrl().

#define PXEBC_DHCP6_BOOT_FILE_URL_PREFIX "tftp://"

Referenced by PxeBcExtractBootFileUrl().

#define PXEBC_DHCP6_C_PORT 546

#define PXEBC_DHCP6_ENTERPRISE_NUM 343

Referenced by PxeBcBuildDhcp6Options(), and PxeBcParseDhcp6Packet().

#define PXEBC_DHCP6_IDX_BOOT_FILE_PARAM 2

Referenced by PxeBcDhcp6BootInfo(), and PxeBcParseDhcp6Packet().

#define PXEBC_DHCP6_IDX_BOOT_FILE_URL 1

Referenced by PxeBcDhcp6BootInfo(), PxeBcHandleDhcp6Offer(), PxeBcParseDhcp6Packet(), PxeBcRetryDhcp6Binl(), and PxeBcSelectDhcp6Offer().

#define PXEBC_DHCP6_IDX_IA_NA 0

Referenced by PxeBcParseDhcp6Packet().

#define PXEBC_DHCP6_IDX_MAX 4

#define PXEBC_DHCP6_IDX_VENDOR_CLASS 3

Referenced by PxeBcParseDhcp6Packet().

#define PXEBC_DHCP6_MAX_BOOT_FILE_SIZE 65535

Referenced by PxeBcExtractBootFileParam().

#define PXEBC_DHCP6_OPT_ARCH 61

Referenced by PxeBcBuildDhcp6Options().

#define PXEBC_DHCP6_OPT_AUTH 11

#define PXEBC_DHCP6_OPT_BOOT_FILE_PARAM 60

Referenced by PxeBcBuildDhcp6Options(), and PxeBcParseDhcp6Packet().

#define PXEBC_DHCP6_OPT_BOOT_FILE_URL 59

Referenced by PxeBcBuildDhcp6Options(), and PxeBcParseDhcp6Packet().

#define PXEBC_DHCP6_OPT_CLIENT_ID 1

#define PXEBC_DHCP6_OPT_ELAPSED_TIME 8

Referenced by PxeBcRequestBootService().

#define PXEBC_DHCP6_OPT_IA_NA 3

Referenced by PxeBcParseDhcp6Packet().

#define PXEBC_DHCP6_OPT_IA_TA 4

#define PXEBC_DHCP6_OPT_IAADDR 5

#define PXEBC_DHCP6_OPT_INTERFACE_ID 18

#define PXEBC_DHCP6_OPT_ORO 6

Referenced by PxeBcBuildDhcp6Options().

#define PXEBC_DHCP6_OPT_PREFERENCE 7

#define PXEBC_DHCP6_OPT_RAPID_COMMIT 14

#define PXEBC_DHCP6_OPT_RECONFIG_ACCEPT 20

#define PXEBC_DHCP6_OPT_RECONFIG_MSG 19

#define PXEBC_DHCP6_OPT_REPLAY_MSG 9

#define PXEBC_DHCP6_OPT_SERVER_ID 2

Referenced by PxeBcRequestBootService().

#define PXEBC_DHCP6_OPT_STATUS_CODE 13

Referenced by PxeBcParseDhcp6Packet().

#define PXEBC_DHCP6_OPT_UNDI 62

Referenced by PxeBcBuildDhcp6Options().

#define PXEBC_DHCP6_OPT_UNICAST 12

#define PXEBC_DHCP6_OPT_USER_CLASS 15

#define PXEBC_DHCP6_OPT_VENDOR_CLASS 16

Referenced by PxeBcBuildDhcp6Options(), and PxeBcParseDhcp6Packet().

#define PXEBC_DHCP6_OPT_VENDOR_OPTS 17

#define PXEBC_DHCP6_OPTION_MAX_NUM 16

Referenced by PxeBcDhcp6Sarr().

#define PXEBC_DHCP6_OPTION_MAX_SIZE 312

Referenced by PxeBcDhcp6Sarr().

#define PXEBC_DHCP6_PACKET_MAX_SIZE 1472

Referenced by EfiPxeBcStart().

#define PXEBC_DHCP6_S_PORT 547

#define PXEBC_IP6_POLICY_MAX 0xff

Referenced by EfiPxeBcStart(), PxeBcRegisterIp6Address(), PxeBcSetIp6Policy(), and PxeBcUnregisterIp6Address().

#define PXEBC_IP6_ROUTE_TABLE_TIMEOUT 10

Referenced by PxeBcRegisterIp6Address().

#define PXEBC_TFTP_URL_SEPARATOR '/'

Referenced by PxeBcExtractBootFileUrl().

Function Documentation

```
EFI_STATUS PxeBcDhcp6Discover ( IN PXEBC_PRIVATE_DATA *   Private,
                                IN UINT16                 Type,
                                IN UINT16 *               Layer,
                                IN BOOLEAN                UseBis,
                                IN EFI_IP_ADDRESS *       DestIp
                              )
```

Build and send out the request packet for the bootfile, and parse the reply.

Parameters:

[in] *Private* The pointer to the PxeBc private data.
[in] *Type* PxeBc option boot item type.
[in] *Layer* The pointer to the option boot item layer.
[in] *UseBis* Use BIS or not.
[in] *DestIp* The pointer to the server address.

Return values:

EFI_SUCCESS	Successfully discovered theboot file.
EFI_OUT_OF_RESOURCES	Failed to allocate resource.
EFI_NOT_FOUND	Can't get the PXE reply packet.
Others	Failed to discover boot file.

Build and send out the request packet for the bootfile, and parse the reply.

Parameters:

[in] *Private* The pointer to PxeBc private data.
[in] *Type* PxeBc option boot item type.
[in] *Layer* The pointer to option boot item layer.
[in] *UseBis* Use BIS or not.
[in] *DestIp* The pointer to the server address.

Return values:

EFI_SUCCESS	Successfully discovered the boot file.
EFI_OUT_OF_RESOURCES	Failed to allocate resources.
EFI_NOT_FOUND	Can't get the PXE reply packet.
Others	Failed to discover the boot file.

References PXEBC_BS_DISCOVER_PORT.

Referenced by PxeBcDiscoverBootServer().

```
EFI_STATUS PxeBcDhcp6Sarr ( IN PXEBC_PRIVATE_DATA *   Private,
                            IN EFI_DHCP6_PROTOCOL *   Dhcp6
                          )
```

Start the DHCPv6 S.A.R.R. process to acquire the IPv6 address and other PXE boot information.

Parameters:

[in] *Private* The pointer to the PxeBc private data.
[in] *Dhcp6* The pointer to EFI_DHCP6_PROTOCOL.

Return values:

EFI_SUCCESS	The S.A.R.R. process successfully finished.
Others	Failed to finish the S.A.R.R. process.

Start the DHCPv6 S.A.R.R. process to acquire the IPv6 address and other PXE boot information.

Parameters:

[in] *Private* The pointer to PxeBc private data.
[in] *Dhcp6* The pointer to the EFI_DHCP6_PROTOCOL

Return values:

EFI_SUCCESS The S.A.R.R. process successfully finished.
Others Failed to finish the S.A.R.R. process.

References PXEBC_DAD_ADDITIONAL_DELAY, PXEBC_DHCP6_OPTION_MAX_NUM, PXEBC_DHCP6_OPTION_MAX_SIZE, PxeBcBuildDhcp6Options(), PxeBcDhcp6CallBack(), and PxeBcHandleDhcp6Offer().

Referenced by EfiPxeBcDhcp().

```
EFI_STATUS PxeBcExtractBootFileParam ( IN CHAR8 *      BootFilePara,
                                       OUT UINT16 *    BootFileSize
                                     )
```

Parse the Boot File Parameter option.

Parameters:

`[in]` *BootFilePara* The pointer to the boot file parameter option data.
`[out]` *BootFileSize* The pointer to the parsed boot file size.

Return values:

EFI_SUCCESS Successfully obtained the boot file size from parameter option.
EFI_NOT_FOUND Failed to extract the boot file size from parameter option.

Parse the Boot File Parameter option.

Parameters:

`[in]` *BootFilePara* The pointer to boot file parameter option data.
`[out]` *BootFileSize* The pointer to the parsed boot file size.

Return values:

EFI_SUCCESS Successfully obtained the boot file size from parameter option.
EFI_NOT_FOUND Failed to extract the boot file size from parameter option.

References PXEBC_DHCP6_MAX_BOOT_FILE_SIZE, and PxeBcUniHexToUint8().

Referenced by PxeBcDhcp6BootInfo().

```
EFI_STATUS PxeBcExtractBootFileUrl ( OUT UINT8 **                FileName,
                                     IN OUT EFI_IPv6_ADDRESS *   SrvAddr,
                                     IN CHAR8 *                  BootFile,
                                     IN UINT16                   Length
                                   )
```

Parse the Boot File URL option.

Parameters:

`[out]` *FileName* The pointer to the boot file name.
`[in,out]` *SrvAddr* The pointer to the boot server address.
`[in]` *BootFile* The pointer to the boot file URL option data.
`[in]` *Length* Length of the boot file URL option data.

Return values:

EFI_ABORTED	User canceled the operation.
EFI_SUCCESS	Selected the boot menu successfully.
EFI_NOT_READY	Read the input key from the keybroad has not finish.

Parse the Boot File URL option.

Parameters:

[out]	*FileName*	The pointer to the boot file name.
[in,out]	*SrvAddr*	The pointer to the boot server address.
[in]	*BootFile*	The pointer to the boot file URL option data.
[in]	*Length*	The length of the boot file URL option data.

Return values:

EFI_ABORTED	User cancel operation.
EFI_SUCCESS	Selected the boot menu successfully.
EFI_NOT_READY	Read the input key from the keybroad has not finish.

References PXEBC_ADDR_END_DELIMITER, PXEBC_ADDR_START_DELIMITER, PXEBC_DHCP6_BOOT_FILE_URL_PREFIX, and PXEBC_TFTP_URL_SEPARATOR.

Referenced by PxeBcDhcp6BootInfo(), and PxeBcRetryDhcp6Binl().

VOID PxeBcFreeBootFileOption (IN LIST_ENTRY * *Head*)

Free all the nodes in the boot file list.

Parameters:

[in] *Head* The pointer to the head of the list.

Free all the nodes in the list for boot file.

Parameters:

[in] *Head* The pointer to the head of list.

EFI_STATUS PxeBcParseDhcp6Packet (IN PXEBC_DHCP6_PACKET_CACHE * *Cache6*)

Parse the cached DHCPv6 packet, including all the options.

Parameters:

[in] *Cache6* The pointer to a cached DHCPv6 packet.

Return values:

EFI_SUCCESS	Parsed the DHCPv6 packet successfully.
EFI_DEVICE_ERROR	Failed to parse and invalid packet.

Parse the cached DHCPv6 packet, including all the options.

Parameters:

[in] *Cache6* The pointer to a cached DHCPv6 packet.

Return values:

EFI_SUCCESS	Parsed the DHCPv6 packet successfully.

EFI_DEVICE_ERROR Failed to parse and invalid the packet.

References DEFAULT_CLASS_ID_DATA, GET_DHCP6_OPTION_SIZE, PXEBC_DHCP6_ENTERPRISE_NUM, PXEBC_DHCP6_IDX_BOOT_FILE_PARAM, PXEBC_DHCP6_IDX_BOOT_FILE_URL, PXEBC_DHCP6_IDX_IA_NA, PXEBC_DHCP6_IDX_VENDOR_CLASS, PXEBC_DHCP6_OPT_BOOT_FILE_PARAM, PXEBC_DHCP6_OPT_BOOT_FILE_URL, PXEBC_DHCP6_OPT_IA_NA, PXEBC_DHCP6_OPT_STATUS_CODE, PXEBC_DHCP6_OPT_VENDOR_CLASS, PxeBcParseDhcp6Options(), PxeOfferTypeDhcpBinl, PxeOfferTypeDhcpOnly, and PxeOfferTypeProxyBinl.

Referenced by EfiPxeBcDiscover(), PxeBcCacheDhcp6Offer(), PxeBcCopyDhcp6Ack(), PxeBcCopyDhcp6Proxy(), and PxeBcRetryDhcp6Binl().

```
EFI_STATUS PxeBcRegisterIp6Address ( IN PXEBC_PRIVATE_DATA *   Private,
                                     IN EFI_IPv6_ADDRESS *     Address
                                   )
```

Register the ready address by Ip6Config protocol.

Parameters:

[in] *Private* The pointer to the PxeBc private data.
[in] *Address* The pointer to the ready address.

Return values:

EFI_SUCCESS Registered the address succesfully.
Others Failed to register the address.

Register the ready station address and gateway by Ip6Config protocol.

Parameters:

[in] *Private* The pointer to PXEBC_PRIVATE_DATA.
[in] *Address* The pointer to the ready address.

Return values:

EFI_SUCCESS Registered the address succesfully.
Others Failed to register the address.

References Policy, PXEBC_DAD_ADDITIONAL_DELAY, PXEBC_IP6_POLICY_MAX, PXEBC_IP6_ROUTE_TABLE_TIMEOUT, PxeBcCheckRouteTable(), and PxeBcCommonNotify().

Referenced by EfiPxeBcSetStationIP(), and PxeBcSetIp6Address().

```
EFI_STATUS PxeBcSetIp6Address ( IN PXEBC_PRIVATE_DATA *   Private   )
```

This function will register the station IP address and flush IP instance to start using the new IP address.

Parameters:

[in] *Private* The pointer to PXEBC_PRIVATE_DATA.

Return values:

EFI_SUCCESS The new IP address has been configured successfully.
Others Failed to configure the address.

References PxeBcFlushStationIp(), PxeBcRegisterIp6Address(), PxeBcShowIp6Addr(), and PxeBcUnregisterIp6Address().

Referenced by PxeBcDhcp6BootInfo().

EFI_STATUS PxeBcSetIp6Policy (IN PXEBC_PRIVATE_DATA * *Private*)

Set the IP6 policy to Automatic.

Parameters:

[in] *Private* The pointer to PXEBC_PRIVATE_DATA.

Return values:

EFI_SUCCESS Switch the IP policy succesfully.

Others Unexpect error happened.

References Policy, and PXEBC_IP6_POLICY_MAX.

Referenced by EfiPxeBcStart().

VOID PxeBcUnregisterIp6Address (IN PXEBC_PRIVATE_DATA * *Private*)

Unregister the address by Ip6Config protocol.

Parameters:

[in] *Private* The pointer to the PxeBc private data.

Unregister the address by Ip6Config protocol.

Parameters:

[in] *Private* The pointer to PXEBC_PRIVATE_DATA.

References PXEBC_IP6_POLICY_MAX.

Referenced by EfiPxeBcStop(), and PxeBcSetIp6Address().

NetworkPkg/UefiPxeBcDxe/PxeBcDriver.c File Reference

Functions

EFI_HANDLE PxeBcGetNicByIp4Children (IN EFI_HANDLE ControllerHandle)
EFI_HANDLE PxeBcGetNicByIp6Children (IN EFI_HANDLE ControllerHandle)
VOID PxeBcDestroyIp4Children (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN PXEBC_PRIVATE_DATA *Private)
VOID PxeBcDestroyIp6Children (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN PXEBC_PRIVATE_DATA *Private)
EFI_STATUS PxeBcCheckIpv6Support (IN EFI_HANDLE ControllerHandle, IN PXEBC_PRIVATE_DATA *Private, OUT BOOLEAN *Ipv6Support)
EFI_STATUS PxeBcCreateIp4Children (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN PXEBC_PRIVATE_DATA *Private)
EFI_STATUS PxeBcCreateIp6Children (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN PXEBC_PRIVATE_DATA *Private)
EFI_STATUS EFIAPI PxeBcDriverEntryPoint (IN EFI_HANDLE ImageHandle, IN EFI_SYSTEM_TABLE *SystemTable)
EFI_STATUS EFIAPI PxeBcSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath, IN UINT8 IpVersion)
EFI_STATUS EFIAPI PxeBcStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath, IN UINT8 IpVersion)
EFI_STATUS EFIAPI PxeBcStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer, IN UINT8 IpVersion)
EFI_STATUS EFIAPI PxeBcIp4DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI PxeBcIp4DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI PxeBcIp4DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)
EFI_STATUS EFIAPI PxeBcIp6DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI PxeBcIp6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)
EFI_STATUS EFIAPI PxeBcIp6DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)

Variables

EFI_DRIVER_BINDING_PROTOCOL gPxeBcIp4DriverBinding
EFI_DRIVER_BINDING_PROTOCOL gPxeBcIp6DriverBinding

Detailed Description

Driver Binding functions implementationfor for UefiPxeBc Driver.

(C) Copyright 2014 Hewlett-Packard Development Company, L.P.
Copyright (c) 2007 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS PxeBcCheckIpv6Support (IN EFI_HANDLE *ControllerHandle*, IN PXEBC_PRIVATE_DATA * *Private*, OUT BOOLEAN * *Ipv6Support*)

Check whether UNDI protocol supports IPv6.

Parameters:

[in]	*ControllerHandle*	Controller handle.
[in]	*Private*	Pointer to PXEBC_PRIVATE_DATA.
[out]	*Ipv6Support*	TRUE if UNDI supports IPv6.

Return values:

EFI_SUCCESS	Get the result whether UNDI supports IPv6 by NII or AIP protocol successfully.
EFI_NOT_FOUND	Don't know whether UNDI supports IPv6 since NII or AIP is not available.

Referenced by PxeBcCreateIp6Children().

EFI_STATUS PxeBcCreateIp4Children (*This*,

```
                                    IN
                                    EFI_DRIVER_BINDING_PROTOCOL
                                    *
                                    IN EFI_HANDLE                    ControllerHandle,
                                    IN PXEBC_PRIVATE_DATA *          Private
                                )
```

Create the opened instances based on IPv4.

Parameters:

[in]	*This*	Pointer to EFI_DRIVER_BINDING_PROTOCOL.
[in]	*ControllerHandle*	Handle of the child to destroy.
[in]	*Private*	Handle Pointer to PXEBC_PRIVATE_DATA.

Return values:

EFI_SUCCESS	The instances based on IPv4 were all created successfully.
Others	An unexpected error occurred.

References gLoadFileProtocolTemplate, PXEBC_DEFAULT_LIFETIME, PXEBC_VIRTUAL_NIC_SIGNATURE, and PxeBcDestroyIp4Children().

Referenced by PxeBcStart().

```
                                    IN
EFI_STATUS PxeBcCreateIp6Children (  EFI_DRIVER_BINDING_PROTOCOL    This,
                                    *
                                    IN EFI_HANDLE                    ControllerHandle,
                                    IN PXEBC_PRIVATE_DATA *          Private
                                )
```

Create the opened instances based on IPv6.

Parameters:

[in]	*This*	Pointer to EFI_DRIVER_BINDING_PROTOCOL.
[in]	*ControllerHandle*	Handle of the child to destroy.
[in]	*Private*	Handle Pointer to PXEBC_PRIVATE_DATA.

Return values:

EFI_SUCCESS	The instances based on IPv6 were all created successfully.
Others	An unexpected error occurred.

References gLoadFileProtocolTemplate, PXEBC_DEFAULT_HOPLIMIT, PXEBC_DEFAULT_LIFETIME, PXEBC_VIRTUAL_NIC_SIGNATURE, PxeBcCheckIpv6Support(), and PxeBcDestroyIp6Children().

Referenced by PxeBcStart().

```
VOID PxeBcDestroyIp4Children ( IN EFI_DRIVER_BINDING_PROTOCOL *  This,
                               IN PXEBC_PRIVATE_DATA *           Private
                             )
```

Destroy the opened instances based on IPv4.

Parameters:

[in] *This* Pointer to the EFI_DRIVER_BINDING_PROTOCOL.
[in] *Private* Pointer to PXEBC_PRIVATE_DATA.

Referenced by PxeBcCreateIp4Children(), PxeBcStart(), and PxeBcStop().

VOID PxeBcDestroyIp6Children (IN EFI_DRIVER_BINDING_PROTOCOL * *This*,
IN PXEBC_PRIVATE_DATA * *Private*
)

Destroy the opened instances based on IPv6.

Parameters:

[in] *This* Pointer to the EFI_DRIVER_BINDING_PROTOCOL.
[in] *Private* Pointer to PXEBC_PRIVATE_DATA.

Referenced by PxeBcCreateIp6Children(), PxeBcStart(), and PxeBcStop().

EFI_STATUS EFIAPI PxeBcDriverEntryPoint (IN EFI_HANDLE *ImageHandle*,
IN EFI_SYSTEM_TABLE * *SystemTable*
)

The entry point for UefiPxeBc driver that installs the driver binding and component name protocol on its image.

Parameters:

[in] *ImageHandle* The Image handle of the driver.
[in] *SystemTable* The system table.

Returns:

EFI_SUCCESS

Others

References gPxeBcComponentName, gPxeBcComponentName2, gPxeBcIp4DriverBinding, and gPxeBcIp6DriverBinding.

EFI_HANDLE PxeBcGetNicByIp4Children (IN EFI_HANDLE *ControllerHandle*)

Get the Nic handle using any child handle in the IPv4 stack.

Parameters:

[in] *ControllerHandle* Pointer to child handle over IPv4.

Returns:

NicHandle The pointer to the Nic handle.

Referenced by PxeBcComponentNameGetControllerName(), and PxeBcStop().

EFI_HANDLE PxeBcGetNicByIp6Children (IN EFI_HANDLE *ControllerHandle*)

Get the Nic handle using any child handle in the IPv6 stack.

Parameters:

[in] *ControllerHandle* Pointer to child handle over IPv6.

Returns:

NicHandle The pointer to the Nic handle.

Referenced by PxeBcComponentNameGetControllerName(), and PxeBcStop().

```
EFI_STATUS EFIAPI
PxeBcIp4DriverBindingStart (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     ControllerHandle,
  IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
)
```

Start this driver on ControllerHandle. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Start() it must also follow these calling restrictions.

Parameters:

[in] *This* The pointer to the driver binding protocol.

[in] *ControllerHandle* The handle of device to be started.

[in] *RemainingDevicePath* Optional parameter used to pick a specific child device to be started.

Return values:

EFI_SUCCESS This driver is installed to ControllerHandle.

EFI_ALREADY_STARTED This driver is already running on ControllerHandle.

other This driver does not support this device.

References PxeBcStart().

```
EFI_STATUS EFIAPI
PxeBcIp4DriverBindingStop (
  IN EFI_DRIVER_BINDING_PROTOCOL *  This,
  IN EFI_HANDLE                     ControllerHandle,
  IN UINTN                          NumberOfChildren,
  IN EFI_HANDLE *                   ChildHandleBuffer
)
```

Stop this driver on ControllerHandle. This service is called by the EFI boot service DisconnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. DisconnectController() must follow these calling restrictions. If any other agent wishes to call Stop() it must also follow these calling restrictions.

Parameters:

[in] *This* Protocol instance pointer.

[in] *ControllerHandle* Handle of device to stop driver on

[in] *NumberOfChildren* Number of Handles in ChildHandleBuffer. If number of children is zero stop the entire bus driver.

[in] *ChildHandleBuffer* List of Child Handles to Stop.

Return values:

EFI_SUCCESS This driver is removed ControllerHandle

EFI_DEVICE_ERROR	An unexpected system or network error occurred.
Others	This driver was not removed from this device.

References PxeBcStop().

```
EFI_STATUS EFIAPI
PxeBcIp4DriverBindingSupported
  (
    IN EFI_DRIVER_BINDING_PROTOCOL *  This,
    IN EFI_HANDLE                     ControllerHandle,
    IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
  )
```

Test to see if this driver supports ControllerHandle. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Supported() it must also follow these calling restrictions.

Parameters:

[in]	*This*	The pointer to the driver binding protocol.
[in]	*ControllerHandle*	The handle of device to be tested.
[in]	*RemainingDevicePath*	Optional parameter used to pick a specific child device to be started.

Return values:

EFI_SUCCESS	This driver supports this device.
EFI_UNSUPPORTED	This driver does not support this device.

References PxeBcSupported().

```
EFI_STATUS EFIAPI
PxeBcIp6DriverBindingStart
  (
    IN EFI_DRIVER_BINDING_PROTOCOL *  This,
    IN EFI_HANDLE                     ControllerHandle,
    IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
  )
```

Start this driver on ControllerHandle. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Start() it must also follow these calling restrictions.

Parameters:

[in]	*This*	The pointer to the driver binding protocol.
[in]	*ControllerHandle*	The handle of device to be started.
[in]	*RemainingDevicePath*	Optional parameter used to pick a specific child device to be started.

Return values:

EFI_SUCCESS	This driver is installed to ControllerHandle.
EFI_ALREADY_STARTED	This driver is already running on ControllerHandle.
other	This driver does not support this device.

References PxeBcStart().

EFI_STATUS EFIAPI PxeBcIp6DriverBindingStop	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN UINTN	*NumberOfChildren*,
	IN EFI_HANDLE *	*ChildHandleBuffer*
	)	

Stop this driver on ControllerHandle. This service is called by the EFI boot service DisconnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. DisconnectController() must follow these calling restrictions. If any other agent wishes to call Stop() it must also follow these calling restrictions.

Parameters:

`[in]`	*This*	Protocol instance pointer.
`[in]`	*ControllerHandle*	Handle of device to stop driver on
`[in]`	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If number of children is zero stop the entire bus driver.
`[in]`	*ChildHandleBuffer*	List of Child Handles to Stop.

Return values:

EFI_SUCCESS	This driver is removed ControllerHandle
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
Others	This driver was not removed from this device.

References PxeBcStop().

EFI_STATUS EFIAPI PxeBcIp6DriverBindingSupported	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*
	)	

Test to see if this driver supports ControllerHandle. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Supported() it must also follow these calling restrictions.

Parameters:

`[in]`	*This*	The pointer to the driver binding protocol.
`[in]`	*ControllerHandle*	The handle of device to be tested.
`[in]`	*RemainingDevicePath*	Optional parameter use to pick a specific child device to be started.

Return values:

EFI_SUCCESS	This driver supports this device.
EFI_UNSUPPORTED	This driver does not support this device.

References PxeBcSupported().

EFI_STATUS EFIAPI PxeBcStart	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,

```
                                IN EFI_HANDLE                          ControllerHandle,
                                IN EFI_DEVICE_PATH_PROTOCOL *          RemainingDevicePath,
                                IN UINT8                               IpVersion
                              )
```

Start this driver on ControllerHandle. This is the worker function for PxeBcIp4(6)DriverBindingStart.

Parameters:

[in]	*This*	The pointer to the driver binding protocol.
[in]	*ControllerHandle*	The handle of device to be started.
[in]	*RemainingDevicePath*	Optional parameter used to pick a specific child device to be started.
[in]	*IpVersion*	IP_VERSION_4 or IP_VERSION_6.

Return values:

EFI_SUCCESS	This driver is installed to ControllerHandle.
EFI_ALREADY_STARTED	This driver is already running on ControllerHandle.
other	This driver does not support this device.

References _PXEBC_PRIVATE_DATA::Controller, _PXEBC_PRIVATE_DATA::DevicePath, gPxeBcProtocolTemplate, _PXEBC_PRIVATE_DATA::Id, _PXEBC_PRIVATE_DATA::Image, _PXEBC_PRIVATE_DATA::Mode, _PXEBC_PRIVATE_DATA::Nii, _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_PRIVATE_DATA_FROM_ID, PXEBC_PRIVATE_DATA_SIGNATURE, PxeBcCreateIp4Children(), PxeBcCreateIp6Children(), PxeBcDestroyIp4Children(), PxeBcDestroyIp6Children(), _PXEBC_PRIVATE_DATA::Signature, and _PXEBC_PRIVATE_DATA::Snp.

Referenced by PxeBcIp4DriverBindingStart(), and PxeBcIp6DriverBindingStart().

```
EFI_STATUS EFIAPI PxeBcStop  ( IN EFI_DRIVER_BINDING_PROTOCOL *  This,
                               IN EFI_HANDLE                     ControllerHandle,
                               IN UINTN                          NumberOfChildren,
                               IN EFI_HANDLE *                   ChildHandleBuffer,
                               IN UINT8                          IpVersion
                             )
```

Stop this driver on ControllerHandle. This is the worker function for PxeBcIp4(6)DriverBindingStop.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to stop driver on.
[in]	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If number of children is zero stop the entire bus driver.
[in]	*ChildHandleBuffer*	List of Child Handles to Stop.
[in]	*IpVersion*	IP_VERSION_4 or IP_VERSION_6.

Return values:

EFI_SUCCESS	This driver was removed ControllerHandle.
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
Others	This driver was not removed from this device

References _PXEBC_PRIVATE_DATA::Controller, _PXEBC_PRIVATE_DATA::Id, _PXEBC_PRIVATE_DATA::Ip4Nic, _PXEBC_PRIVATE_DATA::Ip6Nic,

_PXEBC_VIRTUAL_NIC::Private, _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_PRIVATE_DATA_FROM_ID, PXEBC_VIRTUAL_NIC_FROM_LOADFILE, PxeBcDestroyIp4Children(), PxeBcDestroyIp6Children(), PxeBcGetNicByIp4Children(), and PxeBcGetNicByIp6Children().

Referenced by PxeBcIp4DriverBindingStop(), and PxeBcIp6DriverBindingStop().

EFI_STATUS EFIAPI PxeBcSupported	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*,
	IN UINT8	*IpVersion*
	)	

Test to see if this driver supports ControllerHandle. This is the worker function for PxeBcIp4(6)DriverBindingSupported.

Parameters:

[in]	*This*	The pointer to the driver binding protocol.
[in]	*ControllerHandle*	The handle of device to be tested.
[in]	*RemainingDevicePath*	Optional parameter used to pick a specific child device to be started.
[in]	*IpVersion*	IP_VERSION_4 or IP_VERSION_6.

Return values:

EFI_SUCCESS	This driver supports this device.
EFI_UNSUPPORTED	This driver does not support this device.

Referenced by PxeBcIp4DriverBindingSupported(), and PxeBcIp6DriverBindingSupported().

Variable Documentation

EFI_DRIVER_BINDING_PROTOCOL gPxeBcIp4DriverBinding

Initial value:

```
{
 PxeBcIp4DriverBindingSupported,
 PxeBcIp4DriverBindingStart,
 PxeBcIp4DriverBindingStop,
 0xa,
 NULL,
 NULL
}
```

Referenced by PxeBcDriverEntryPoint().

EFI_DRIVER_BINDING_PROTOCOL gPxeBcIp6DriverBinding

Initial value:

```
{
 PxeBcIp6DriverBindingSupported,
 PxeBcIp6DriverBindingStart,
 PxeBcIp6DriverBindingStop,
 0xa,
 NULL,
 NULL
```

```
}
```

Referenced by PxeBcDriverEntryPoint().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/UefiPxeBcDxe/PxeBcDriver.h File Reference

Functions

EFI_STATUS EFIAPI PxeBcIp4DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI PxeBcIp4DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI PxeBcIp4DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)

EFI_STATUS EFIAPI PxeBcIp6DriverBindingSupported (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI PxeBcIp6DriverBindingStart (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN EFI_DEVICE_PATH_PROTOCOL *RemainingDevicePath)

EFI_STATUS EFIAPI PxeBcIp6DriverBindingStop (IN EFI_DRIVER_BINDING_PROTOCOL *This, IN EFI_HANDLE ControllerHandle, IN UINTN NumberOfChildren, IN EFI_HANDLE *ChildHandleBuffer)

Variables

EFI_COMPONENT_NAME_PROTOCOL gPxeBcComponentName

EFI_COMPONENT_NAME2_PROTOCOL gPxeBcComponentName2

Detailed Description

Driver Binding functions declaration for UefiPxeBc Driver.

Copyright (c) 2007 - 2013, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

EFI_STATUS EFIAPI PxeBcIp4DriverBindingStart	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*
	)	

Start this driver on ControllerHandle. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Start() it must also follow these calling restrictions.

Parameters:

[in]	*This*	The pointer to the driver binding protocol.
[in]	*ControllerHandle*	The handle of device to be started.
[in]	*RemainingDevicePath*	Optional parameter used to pick a specific child device to be started.

Return values:

EFI_SUCCESS	This driver is installed to ControllerHandle.
EFI_ALREADY_STARTED	This driver is already running on ControllerHandle.
other	This driver does not support this device.

References PxeBcStart().

EFI_STATUS EFIAPI PxeBcIp4DriverBindingStop	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN UINTN	*NumberOfChildren*,
	IN EFI_HANDLE *	*ChildHandleBuffer*
	)	

Stop this driver on ControllerHandle. This service is called by the EFI boot service DisconnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. DisconnectController() must follow these calling restrictions. If any other agent wishes to call Stop() it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to stop driver on
[in]	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If number of children is zero stop the entire bus driver.
[in]	*ChildHandleBuffer*	List of Child Handles to Stop.

Return values:

EFI_SUCCESS	This driver is removed ControllerHandle
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
Others	This driver was not removed from this device.

References PxeBcStop().

EFI_STATUS EFIAPI PxeBcIp4DriverBindingSupported	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*
	)	

Test to see if this driver supports ControllerHandle. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Supported() it must also follow these calling restrictions.

Parameters:

[in]	*This*	The pointer to the driver binding protocol.
[in]	*ControllerHandle*	The handle of device to be tested.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to be started.

Return values:

EFI_SUCCESS	This driver supports this device.
EFI_UNSUPPORTED	This driver does not support this device.

Test to see if this driver supports ControllerHandle. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Supported() it must also follow these calling restrictions.

Parameters:

[in]	*This*	The pointer to the driver binding protocol.
[in]	*ControllerHandle*	The handle of device to be tested.
[in]	*RemainingDevicePath*	Optional parameter used to pick a specific child device to be started.

Return values:

EFI_SUCCESS	This driver supports this device.
EFI_UNSUPPORTED	This driver does not support this device.

References PxeBcSupported().

EFI_STATUS EFIAPI PxeBcIp6DriverBindingStart	(IN EFI_DRIVER_BINDING_PROTOCOL *	*This*,
	IN EFI_HANDLE	*ControllerHandle*,
	IN EFI_DEVICE_PATH_PROTOCOL *	*RemainingDevicePath*
	)	

Start this driver on ControllerHandle. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Start() it must also follow these calling restrictions.

Parameters:

[in]	*This*	The pointer to the driver binding protocol.
[in]	*ControllerHandle*	The handle of device to be started.
[in]	*RemainingDevicePath*	Optional parameter used to pick a specific child device to be started.

Return values:

EFI_SUCCESS	This driver is installed to ControllerHandle.
EFI_ALREADY_STARTED	This driver is already running on ControllerHandle.
other	This driver does not support this device.

References PxeBcStart().

```
EFI_STATUS EFIAPI
PxeBcIp6DriverBindingStop (
    IN EFI_DRIVER_BINDING_PROTOCOL *  This,
    IN EFI_HANDLE                     ControllerHandle,
    IN UINTN                          NumberOfChildren,
    IN EFI_HANDLE *                   ChildHandleBuffer
)
```

Stop this driver on ControllerHandle. This service is called by the EFI boot service DisconnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. DisconnectController() must follow these calling restrictions. If any other agent wishes to call Stop() it must also follow these calling restrictions.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*ControllerHandle*	Handle of device to stop driver on
[in]	*NumberOfChildren*	Number of Handles in ChildHandleBuffer. If number of children is zero stop the entire bus driver.
[in]	*ChildHandleBuffer*	List of Child Handles to Stop.

Return values:

EFI_SUCCESS	This driver is removed ControllerHandle
EFI_DEVICE_ERROR	An unexpected system or network error occurred.
Others	This driver was not removed from this device.

References PxeBcStop().

```
EFI_STATUS EFIAPI
PxeBcIp6DriverBindingSupported (
    IN EFI_DRIVER_BINDING_PROTOCOL *  This,
    IN EFI_HANDLE                     ControllerHandle,
    IN EFI_DEVICE_PATH_PROTOCOL *     RemainingDevicePath
)
```

Test to see if this driver supports ControllerHandle. This service is called by the EFI boot service ConnectController(). In order to make drivers as small as possible, there are a few calling restrictions for this service. ConnectController() must follow these calling restrictions. If any other agent wishes to call Supported() it must also follow these calling restrictions.

Parameters:

[in]	*This*	The pointer to the driver binding protocol.
[in]	*ControllerHandle*	The handle of device to be tested.
[in]	*RemainingDevicePath*	Optional parameter use to pick a specific child device to be started.

Return values:

EFI_SUCCESS	This driver supports this device.

EFI_UNSUPPORTED This driver does not support this device.

References PxeBcSupported().

Variable Documentation

EFI_COMPONENT_NAME_PROTOCOL gPxeBcComponentName

Referenced by PxeBcComponentNameGetControllerName(), PxeBcComponentNameGetDriverName(), and PxeBcDriverEntryPoint().

EFI_COMPONENT_NAME2_PROTOCOL gPxeBcComponentName2

Referenced by PxeBcDriverEntryPoint().

NetworkPkg/UefiPxeBcDxe/PxeBcImpl.c File Reference

Functions

EFI_STATUS EFIAPI EfiPxeBcStart (IN EFI_PXE_BASE_CODE_PROTOCOL *This, IN BOOLEAN UseIpv6)

EFI_STATUS EFIAPI EfiPxeBcStop (IN EFI_PXE_BASE_CODE_PROTOCOL *This)

EFI_STATUS EFIAPI EfiPxeBcDhcp (IN EFI_PXE_BASE_CODE_PROTOCOL *This, IN BOOLEAN SortOffers)

EFI_STATUS EFIAPI EfiPxeBcDiscover (IN EFI_PXE_BASE_CODE_PROTOCOL *This, IN UINT16 Type, IN UINT16 *Layer, IN BOOLEAN UseBis, IN EFI_PXE_BASE_CODE_DISCOVER_INFO *Info)

EFI_STATUS EFIAPI EfiPxeBcMtftp (IN EFI_PXE_BASE_CODE_PROTOCOL *This, IN EFI_PXE_BASE_CODE_TFTP_OPCODE Operation, IN OUT VOID *BufferPtr, IN BOOLEAN Overwrite, IN OUT UINT64 *BufferSize, IN UINTN *BlockSize, IN EFI_IP_ADDRESS *ServerIp, IN UINT8 *Filename, IN EFI_PXE_BASE_CODE_MTFTP_INFO *Info, IN BOOLEAN DontUseBuffer)

EFI_STATUS EFIAPI EfiPxeBcUdpWrite (IN EFI_PXE_BASE_CODE_PROTOCOL *This, IN UINT16 OpFlags, IN EFI_IP_ADDRESS *DestIp, IN EFI_PXE_BASE_CODE_UDP_PORT *DestPort, IN EFI_IP_ADDRESS *GatewayIp, IN EFI_IP_ADDRESS *SrcIp, IN OUT EFI_PXE_BASE_CODE_UDP_PORT *SrcPort, IN UINTN *HeaderSize, IN VOID *HeaderPtr, IN UINTN *BufferSize, IN VOID *BufferPtr)

EFI_STATUS EFIAPI EfiPxeBcUdpRead (IN EFI_PXE_BASE_CODE_PROTOCOL *This, IN UINT16 OpFlags, IN OUT EFI_IP_ADDRESS *DestIp, IN OUT EFI_PXE_BASE_CODE_UDP_PORT *DestPort, IN OUT EFI_IP_ADDRESS *SrcIp, IN OUT EFI_PXE_BASE_CODE_UDP_PORT *SrcPort, IN UINTN *HeaderSize, IN VOID *HeaderPtr, IN OUT UINTN *BufferSize, IN VOID *BufferPtr)

EFI_STATUS EFIAPI EfiPxeBcSetIpFilter (IN EFI_PXE_BASE_CODE_PROTOCOL *This, IN EFI_PXE_BASE_CODE_IP_FILTER *NewFilter)

EFI_STATUS EFIAPI EfiPxeBcArp (IN EFI_PXE_BASE_CODE_PROTOCOL *This, IN EFI_IP_ADDRESS *IpAddr, IN

EFI_MAC_ADDRESS *MacAddr)
EFI_STATUS EFIAPI EfiPxeBcSetParameters (IN EFI_PXE_BASE_CODE_PROTOCOL *This, IN BOOLEAN *NewAutoArp, IN BOOLEAN *NewSendGUID, IN UINT8 *NewTTL, IN UINT8 *NewToS, IN BOOLEAN *NewMakeCallback)
EFI_STATUS EFIAPI EfiPxeBcSetStationIP (IN EFI_PXE_BASE_CODE_PROTOCOL *This, IN EFI_IP_ADDRESS *NewStationIp, IN EFI_IP_ADDRESS *NewSubnetMask)
EFI_STATUS EFIAPI EfiPxeBcSetPackets (IN EFI_PXE_BASE_CODE_PROTOCOL *This, IN BOOLEAN *NewDhcpDiscoverValid, IN BOOLEAN *NewDhcpAckReceived, IN BOOLEAN *NewProxyOfferReceived, IN BOOLEAN *NewPxeDiscoverValid, IN BOOLEAN *NewPxeReplyReceived, IN BOOLEAN *NewPxeBisReplyReceived, IN EFI_PXE_BASE_CODE_PACKET *NewDhcpDiscover, IN EFI_PXE_BASE_CODE_PACKET *NewDhcpAck, IN EFI_PXE_BASE_CODE_PACKET *NewProxyOffer, IN EFI_PXE_BASE_CODE_PACKET *NewPxeDiscover, IN EFI_PXE_BASE_CODE_PACKET *NewPxeReply, IN EFI_PXE_BASE_CODE_PACKET *NewPxeBisReply)
EFI_PXE_BASE_CODE_CALLBACK_STATUS EFIAPI EfiPxeLoadFileCallback (IN EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL *This, IN EFI_PXE_BASE_CODE_FUNCTION Function, IN BOOLEAN Received, IN UINT32 PacketLength, IN EFI_PXE_BASE_CODE_PACKET *PacketPtr)
EFI_STATUS EFIAPI EfiPxeLoadFile (IN EFI_LOAD_FILE_PROTOCOL *This, IN EFI_DEVICE_PATH_PROTOCOL *FilePath, IN BOOLEAN BootPolicy, IN OUT UINTN *BufferSize, IN VOID *Buffer)

Variables

EFI_PXE_BASE_CODE_PROTOCOL gPxeBcProtocolTemplate
EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL gPxeBcCallBackTemplate
EFI_LOAD_FILE_PROTOCOL gLoadFileProtocolTemplate = { EfiPxeLoadFile }

Detailed Description

This implementation of EFI_PXE_BASE_CODE_PROTOCOL and EFI_LOAD_FILE_PROTOCOL.

Copyright (c) 2007 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI EfiPxeBcArp ( IN EFI_PXE_BASE_CODE_PROTOCOL *  This,
                                IN EFI_IP_ADDRESS *              IpAddr,
                                IN EFI_MAC_ADDRESS *             MacAddr
                              )
```

Uses the ARP protocol to resolve a MAC address. It is not supported for IPv6.

This function uses the ARP protocol to resolve a MAC address. The IP address specified by IpAddr is used to resolve a MAC address. If the ARP protocol succeeds in resolving the specified address, then the ArpCacheEntries and ArpCache fields of the mode data are updated, and EFI_SUCCESS is returned. If MacAddr is not NULL, the resolved MAC address is placed there as well. If the PXE Base Code protocol is in the stopped state, then EFI_NOT_STARTED is returned. If the ARP protocol encounters a timeout condition while attempting to resolve an address, then EFI_TIMEOUT is returned. If the Callback Protocol does not return EFI_PXE_BASE_CODE_CALLBACK_STATUS_CONTINUE, then EFI_ABORTED is returned.

Parameters:

[in]	*This*	Pointer to the EFI_PXE_BASE_CODE_PROTOCOL instance.
[in]	*IpAddr*	Pointer to the IP address that is used to resolve a MAC address.
[in]	*MacAddr*	If not NULL, a pointer to the MAC address that was resolved with the ARP protocol.

Return values:

EFI_SUCCESS	The IP or MAC address was resolved.
EFI_NOT_STARTED	The PXE Base Code Protocol is in the stopped state.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
EFI_ICMP_ERROR	An error occur with the ICMP packet message.

References _PXEBC_PRIVATE_DATA::Arp, _PXEBC_PRIVATE_DATA::IsAddressOk, _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_PRIVATE_DATA_FROM_PXEBC, PxeBcArpCacheUpdate(), PxeBcCheckArpCache(), and PxeBcCommonNotify().

```
EFI_STATUS EFIAPI EfiPxeBcDhcp ( IN EFI_PXE_BASE_CODE_PROTOCOL *  This,
                                 IN BOOLEAN                       SortOffers
                               )
```

Attempts to complete a DHCPv4 D.O.R.A. (discover / offer / request / acknowledge) or DHCPv6 S.A.R.R (solicit / advertise / request / reply) sequence.

If SortOffers is TRUE, then the cached DHCP offer packets will be sorted before they are tried. If SortOffers is FALSE, then the cached DHCP offer packets will be tried in the order in which they are received. Please see the Preboot Execution Environment (PXE) Specification and Unified Extensible Firmware Interface (UEFI) Specification for additional details on the implementation of DHCP. If the Callback Protocol does not

return EFI_PXE_BASE_CODE_CALLBACK_STATUS_CONTINUE, then the DHCP sequence will be stopped and EFI_ABORTED will be returned.

Parameters:

[in]	*This*	Pointer to the EFI_PXE_BASE_CODE_PROTOCOL instance.
[in]	*SortOffers*	TRUE if the offers received should be sorted. Set to FALSE to try the offers in the order that they are received.

Return values:

EFI_SUCCESS	Valid DHCP has completed.
EFI_NOT_STARTED	The PXE Base Code Protocol is in the stopped state.
EFI_INVALID_PARAMETER	The This parameter is NULL or does not point to a valid EFI_PXE_BASE_CODE_PROTOCOL structure.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
EFI_OUT_OF_RESOURCES	Could not allocate enough memory to complete the DHCP Protocol.
EFI_ABORTED	The callback function aborted the DHCP Protocol.
EFI_TIMEOUT	The DHCP Protocol timed out.
EFI_ICMP_ERROR	An ICMP error packet was received during the DHCP session.
EFI_NO_RESPONSE	Valid PXE offer was not received.

References _PXEBC_PRIVATE_DATA::Dhcp4, _PXEBC_PRIVATE_DATA::Dhcp6, _PXEBC_PRIVATE_DATA::ElapsedTime, _PXEBC_PRIVATE_DATA::Function, _PXEBC_PRIVATE_DATA::IsOfferSorted, _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_PRIVATE_DATA_FROM_PXEBC, PxeBcDhcp4Dora(), PxeBcDhcp6Sarr(), _PXEBC_PRIVATE_DATA::SolicitTimes, _PXEBC_PRIVATE_DATA::Udp4CfgData, _PXEBC_PRIVATE_DATA::Udp4Read, _PXEBC_PRIVATE_DATA::Udp6CfgData, and _PXEBC_PRIVATE_DATA::Udp6Read.

```
EFI_STATUS EFIAPI EfiPxeBcDiscover ( IN EFI_PXE_BASE_CODE_PROTOCOL *          This,
                                     IN UINT16                                Type,
                                     IN UINT16 *                              Layer,
                                     IN BOOLEAN                               UseBis,
                                     IN EFI_PXE_BASE_CODE_DISCOVER_INFO *     Info
                                   )
```

Attempts to complete the PXE Boot Server and/or boot image discovery sequence.

This function attempts to complete the PXE Boot Server and/or boot image discovery sequence. If this sequence is completed, then EFI_SUCCESS is returned, and the PxeDiscoverValid, PxeDiscover, PxeReplyReceived, and PxeReply fields of the EFI_PXE_BASE_CODE_MODE structure are filled in. If UseBis is TRUE, then the PxeBisReplyReceived and PxeBisReply fields of the EFI_PXE_BASE_CODE_MODE structure will also be filled in. If UseBis is FALSE, then PxeBisReplyValid will be set to FALSE. In the structure referenced by parameter Info, the PXE Boot Server list, SrvList[], has two uses: It is the Boot Server IP address list used for unicast discovery (if the UseUCast field is TRUE), and it is the list used for Boot Server verification (if the MustUseList field is TRUE). Also, if the MustUseList field in that structure is TRUE and the AcceptAnyResponse field in the SrvList[] array is TRUE, any Boot Server reply of that type will be accepted. If the AcceptAnyResponse field is FALSE, only responses from Boot Servers with matching IP addresses will be accepted. This function can take at least 10 seconds to timeout and return control to the caller. If the Discovery sequence does not complete, then EFI_TIMEOUT will be returned. Please see the Preboot Execution Environment (PXE) Specification for additional details on the implementation of the Discovery sequence. If the Callback Protocol does not return EFI_PXE_BASE_CODE_CALLBACK_STATUS_CONTINUE, then the Discovery sequence is stopped and

EFI_ABORTED will be returned.

Parameters:

[in]	*This*	Pointer to the EFI_PXE_BASE_CODE_PROTOCOL instance.
[in]	*Type*	The type of bootstrap to perform.
[in]	*Layer*	Pointer to the boot server layer number to discover, which must be PXE_BOOT_LAYER_INITIAL when a new server type is being discovered.
[in]	*UseBis*	TRUE if Boot Integrity Services are to be used. FALSE otherwise.
[in]	*Info*	Pointer to a data structure that contains additional information on the type of discovery operation that is to be performed. It is optional.

Return values:

EFI_SUCCESS	The Discovery sequence has been completed.
EFI_NOT_STARTED	The PXE Base Code Protocol is in the stopped state.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
EFI_OUT_OF_RESOURCES	Could not allocate enough memory to complete Discovery.
EFI_ABORTED	The callback function aborted the Discovery sequence.
EFI_TIMEOUT	The Discovery sequence timed out.
EFI_ICMP_ERROR	An ICMP error packet was received during the PXE discovery session.

References PXEBC_DHCP4_PACKET::Ack, PXEBC_DHCP6_PACKET::Ack, PXEBC_DHCP_PACKET_CACHE::Dhcp4, PXEBC_DHCP_PACKET_CACHE::Dhcp6, _PXEBC_PRIVATE_DATA::Function, PXEBC_BOOT_SVR_ENTRY::IpAddr, _PXEBC_PRIVATE_DATA::IsAddressOk, _PXEBC_PRIVATE_DATA::IsDoDiscover, PXEBC_DHCP4_PACKET_CACHE::Packet, PXEBC_DHCP6_PACKET_CACHE::Packet, _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_PRIVATE_DATA_FROM_PXEBC, PxeBcDiscoverBootServer(), PxeBcExtractDiscoverInfo(), PxeBcParseDhcp4Packet(), PxeBcParseDhcp6Packet(), _PXEBC_PRIVATE_DATA::PxeReply, _PXEBC_PRIVATE_DATA::ServerIp, _PXEBC_PRIVATE_DATA::Udp4CfgData, _PXEBC_PRIVATE_DATA::Udp4Read, _PXEBC_PRIVATE_DATA::Udp6CfgData, and _PXEBC_PRIVATE_DATA::Udp6Read.

```
EFI_STATUS EFIAPI EfiPxeBcMtftp ( IN EFI_PXE_BASE_CODE_PROTOCOL *      This,
                                  IN EFI_PXE_BASE_CODE_TFTP_OPCODE     Operation,
                                  IN OUT VOID *                        BufferPtr,
                                  IN BOOLEAN                           Overwrite,
                                  IN OUT UINT64 *                      BufferSize,
                                  IN UINTN *                           BlockSize,
                                  IN EFI_IP_ADDRESS *                  ServerIp,
                                  IN UINT8 *                           Filename,
                                  IN EFI_PXE_BASE_CODE_MTFTP_INFO *    Info,
                                  IN BOOLEAN                           DontUseBuffer
                                )
```

Used to perform TFTP and MTFTP services.

This function is used to perform TFTP and MTFTP services. This includes the TFTP operations to get the size of a file, read a directory, read a file, and write a file. It also includes the MTFTP operations to get the size of a file, read a directory, and read a file. The type of operation is specified by Operation. If the callback function that is invoked during the TFTP/MTFTP operation does not return

EFI_PXE_BASE_CODE_CALLBACK_STATUS_CONTINUE, then EFI_ABORTED will be returned. For read operations, the return data will be placed in the buffer specified by BufferPtr. If BufferSize is too small to contain the entire downloaded file, then EFI_BUFFER_TOO_SMALL will be returned and BufferSize will be set to zero, or the size of the requested file. (NOTE: the size of the requested file is only returned if the TFTP server supports TFTP options). If BufferSize is large enough for the read operation, then BufferSize will be set to the size of the downloaded file, and EFI_SUCCESS will be returned. Applications using the PxeBc.Mtftp() services should use the get-file-size operations to determine the size of the downloaded file prior to using the read-file operations-especially when downloading large (greater than 64 MB) files-instead of making two calls to the read-file operation. Following this recommendation will save time if the file is larger than expected and the TFTP server does not support TFTP option extensions. Without TFTP option extension support, the client must download the entire file, counting and discarding the received packets, to determine the file size. For write operations, the data to be sent is in the buffer specified by BufferPtr. BufferSize specifies the number of bytes to send. If the write operation completes successfully, then EFI_SUCCESS will be returned. For TFTP "get file size" operations, the size of the requested file or directory is returned in BufferSize, and EFI_SUCCESS will be returned. If the TFTP server does not support options, the file will be downloaded into a bit bucket and the length of the downloaded file will be returned. For MTFTP "get file size" operations, if the MTFTP server does not support the "get file size" option, EFI_UNSUPPORTED will be returned. This function can take up to 10 seconds to timeout and return control to the caller. If the TFTP sequence does not complete, EFI_TIMEOUT will be returned. If the Callback Protocol does not return EFI_PXE_BASE_CODE_CALLBACK_STATUS_CONTINUE, then the TFTP sequence is stopped and EFI_ABORTED will be returned.

Parameters:

`[in]`	*This*	Pointer to the EFI_PXE_BASE_CODE_PROTOCOL instance.
`[in]`	*Operation*	The type of operation to perform.
`[in,out]`	*BufferPtr*	A pointer to the data buffer.
`[in]`	*Overwrite*	Only used on write file operations. TRUE if a file on a remote server can be overwritten.
`[in,out]`	*BufferSize*	For get-file-size operations, *BufferSize returns the size of the requested file.
`[in]`	*BlockSize*	The requested block size to be used during a TFTP transfer.
`[in]`	*ServerIp*	The TFTP / MTFTP server IP address.
`[in]`	*Filename*	A Null-terminated ASCII string that specifies a directory name or a file name.
`[in]`	*Info*	Pointer to the MTFTP information.
`[in]`	*DontUseBuffer*	Set to FALSE for normal TFTP and MTFTP read file operation.

Return values:

EFI_SUCCESS	The TFTP/MTFTP operation was completed.
EFI_NOT_STARTED	The PXE Base Code Protocol is in the stopped state.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
EFI_BUFFER_TOO_SMALL	The buffer is not large enough to complete the read operation.
EFI_ABORTED	The callback function aborted the TFTP/MTFTP operation.
EFI_TIMEOUT	The TFTP/MTFTP operation timed out.
EFI_ICMP_ERROR	An ICMP error packet was received during the MTFTP session.
EFI_TFTP_ERROR	A TFTP error packet was received during the MTFTP session.

References _PXEBC_PRIVATE_DATA::GatewayIp, PXE_MTFTP_DEFAULT_BLOCK_SIZE, _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_MTFTP_RETRIES, PXEBC_MTFTP_TIMEOUT, PXEBC_PRIVATE_DATA_FROM_PXEBC, PxeBcTftpGetFileSize(), PxeBcTftpReadDirectory(), PxeBcTftpReadFile(), PxeBcTftpWriteFile(), _PXEBC_PRIVATE_DATA::StationIp, _PXEBC_PRIVATE_DATA::SubnetMask, _PXEBC_PRIVATE_DATA::Udp4CfgData,

_PXEBC_PRIVATE_DATA::Udp4Read, _PXEBC_PRIVATE_DATA::Udp6CfgData, and _PXEBC_PRIVATE_DATA::Udp6Read.

EFI_STATUS EFIAPI EfiPxeBcSetIpFilter (	IN EFI_PXE_BASE_CODE_PROTOCOL *	*This*,
	IN EFI_PXE_BASE_CODE_IP_FILTER *	*NewFilter*
)		

Updates the IP receive filters of a network device and enables software filtering.

The NewFilter field is used to modify the network device's current IP receive filter settings and to enable a software filter. This function updates the IpFilter field of the EFI_PXE_BASE_CODE_MODE structure with the contents of NewIpFilter. The software filter is used when the USE_FILTER in OpFlags is set to UdpRead(). The current hardware filter remains in effect no matter what the settings of OpFlags. This is so that the meaning of ANY_DEST_IP set in OpFlags to UdpRead() is from those packets whose reception is enabled in hardware-physical NIC address (unicast), broadcast address, logical address or addresses (multicast), or all (promiscuous). UdpRead() does not modify the IP filter settings. Dhcp(), Discover(), and Mtftp() set the IP filter, and return with the IP receive filter list emptied and the filter set to EFI_PXE_BASE_CODE_IP_FILTER_STATION_IP. If an application or driver wishes to preserve the IP receive filter settings, it will have to preserve the IP receive filter settings before these calls, and use SetIpFilter() to restore them after the calls. If incompatible filtering is requested (for example, PROMISCUOUS with anything else), or if the device does not support a requested filter setting and it cannot be accommodated in software (for example, PROMISCUOUS not supported), EFI_INVALID_PARAMETER will be returned. The IPlist field is used to enable IPs other than the StationIP. They may be multicast or unicast. If IPcnt is set as well as EFI_PXE_BASE_CODE_IP_FILTER_STATION_IP, then both the StationIP and the IPs from the IPlist will be used.

Parameters:

[in]	*This*	Pointer to the EFI_PXE_BASE_CODE_PROTOCOL instance.
[in]	*NewFilter*	Pointer to the new set of IP receive filters.

Return values:

EFI_SUCCESS	The IP receive filter settings were updated.
EFI_NOT_STARTED	The PXE Base Code Protocol is in the stopped state.
EFI_INVALID_PARAMETER	One or more parameters are invalid.

References _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_PRIVATE_DATA_FROM_PXEBC, _PXEBC_PRIVATE_DATA::Udp4CfgData, _PXEBC_PRIVATE_DATA::Udp4Read, _PXEBC_PRIVATE_DATA::Udp6CfgData, and _PXEBC_PRIVATE_DATA::Udp6Read.

EFI_STATUS EFIAPI EfiPxeBcSetPackets (	IN EFI_PXE_BASE_CODE_PROTOCOL *	*This*,
	IN BOOLEAN *	*NewDhcpDiscoverValid*,
	IN BOOLEAN *	*NewDhcpAckReceived*,
	IN BOOLEAN *	*NewProxyOfferReceived*,
	IN BOOLEAN *	*NewPxeDiscoverValid*,
	IN BOOLEAN *	*NewPxeReplyReceived*,
	IN BOOLEAN *	*NewPxeBisReplyReceived*,
	IN EFI_PXE_BASE_CODE_PACKET *	*NewDhcpDiscover*,
	IN EFI_PXE_BASE_CODE_PACKET *	*NewDhcpAck*,
	IN EFI_PXE_BASE_CODE_PACKET *	*NewProxyOffer*,

```
IN EFI_PXE_BASE_CODE_PACKET *    NewPxeDiscover,
IN EFI_PXE_BASE_CODE_PACKET *    NewPxeReply,
IN EFI_PXE_BASE_CODE_PACKET *    NewPxeBisReply
)
```

Updates the contents of the cached DHCP and Discover packets.

The pointers to the new packets are used to update the contents of the cached packets in the EFI_PXE_BASE_CODE_MODE structure.

Parameters:

[in]	*This*	Pointer to the EFI_PXE_BASE_CODE_PROTOCOL instance.
[in]	*NewDhcpDiscoverValid*	Pointer to a value that will replace the current DhcpDiscoverValid field.
[in]	*NewDhcpAckReceived*	Pointer to a value that will replace the current DhcpAckReceived field.
[in]	*NewProxyOfferReceived*	Pointer to a value that will replace the current ProxyOfferReceived field.
[in]	*NewPxeDiscoverValid*	Pointer to a value that will replace the current ProxyOfferReceived field.
[in]	*NewPxeReplyReceived*	Pointer to a value that will replace the current PxeReplyReceived field.
[in]	*NewPxeBisReplyReceived*	Pointer to a value that will replace the current PxeBisReplyReceived field.
[in]	*NewDhcpDiscover*	Pointer to the new cached DHCP Discover packet contents.
[in]	*NewDhcpAck*	Pointer to the new cached DHCP Ack packet contents.
[in]	*NewProxyOffer*	Pointer to the new cached Proxy Offer packet contents.
[in]	*NewPxeDiscover*	Pointer to the new cached PXE Discover packet contents.
[in]	*NewPxeReply*	Pointer to the new cached PXE Reply packet contents.
[in]	*NewPxeBisReply*	Pointer to the new cached PXE BIS Reply packet contents.

Return values:

EFI_SUCCESS	The cached packet contents were updated.
EFI_NOT_STARTED	The PXE Base Code Protocol is in the stopped state.
EFI_INVALID_PARAMETER	This is NULL or does not point to a valid EFI_PXE_BASE_CODE_PROTOCOL structure.

References _PXEBC_PRIVATE_DATA::PxeBc, and PXEBC_PRIVATE_DATA_FROM_PXEBC.

```
EFI_STATUS EFIAPI
EfiPxeBcSetParameters
(
  IN EFI_PXE_BASE_CODE_PROTOCOL *    This,
  IN BOOLEAN *                       NewAutoArp,
  IN BOOLEAN *                       NewSendGUID,
  IN UINT8 *                         NewTTL,
  IN UINT8 *                         NewToS,
  IN BOOLEAN *                       NewMakeCallback
)
```

Updates the parameters that affect the operation of the PXE Base Code Protocol.

This function sets parameters that affect the operation of the PXE Base Code Protocol. The parameter specified by NewAutoArp is used to control the generation of ARP protocol packets. If NewAutoArp is

TRUE, then ARP Protocol packets will be generated as required by the PXE Base Code Protocol. If NewAutoArp is FALSE, then no ARP Protocol packets will be generated. In this case, the only mappings that are available are those stored in the ArpCache of the EFI_PXE_BASE_CODE_MODE structure. If there are not enough mappings in the ArpCache to perform a PXE Base Code Protocol service, then the service will fail. This function updates the AutoArp field of the EFI_PXE_BASE_CODE_MODE structure to NewAutoArp. The SetParameters() call must be invoked after a Callback Protocol is installed to enable the use of callbacks.

Parameters:

[in]	*This*	Pointer to the EFI_PXE_BASE_CODE_PROTOCOL instance.
[in]	*NewAutoArp*	If not NULL, a pointer to a value that specifies whether to replace the current value of AutoARP.
[in]	*NewSendGUID*	If not NULL, a pointer to a value that specifies whether to replace the current value of SendGUID.
[in]	*NewTTL*	If not NULL, a pointer to be used in place of the current value of TTL, the "time to live" field of the IP header.
[in]	*NewToS*	If not NULL, a pointer to be used in place of the current value of ToS, the "type of service" field of the IP header.
[in]	*NewMakeCallback*	If not NULL, a pointer to a value that specifies whether to replace the current value of the MakeCallback field of the Mode structure.

Return values:

EFI_SUCCESS	The new parameters values were updated.
EFI_NOT_STARTED	The PXE Base Code Protocol is in the stopped state.
EFI_INVALID_PARAMETER	One or more parameters are invalid.

References _PXEBC_PRIVATE_DATA::Controller, _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_PRIVATE_DATA_FROM_PXEBC, and _PXEBC_PRIVATE_DATA::PxeBcCallback.

```
EFI_STATUS EFIAPI
EfiPxeBcSetStationIP
(
  IN EFI_PXE_BASE_CODE_PROTOCOL *   This,
  IN EFI_IP_ADDRESS *               NewStationIp,
  IN EFI_IP_ADDRESS *               NewSubnetMask
)
```

Updates the station IP address and/or subnet mask values of a network device.

This function updates the station IP address and/or subnet mask values of a network device. The NewStationIp field is used to modify the network device's current IP address. If NewStationIP is NULL, then the current IP address will not be modified. Otherwise, this function updates the StationIp field of the EFI_PXE_BASE_CODE_MODE structure with NewStationIp. The NewSubnetMask field is used to modify the network device's current subnet mask. If NewSubnetMask is NULL, then the current subnet mask will not be modified. Otherwise, this function updates the SubnetMask field of the EFI_PXE_BASE_CODE_MODE structure with NewSubnetMask.

Parameters:

[in]	*This*	Pointer to the EFI_PXE_BASE_CODE_PROTOCOL instance.
[in]	*NewStationIp*	Pointer to the new IP address to be used by the network device.
[in]	*NewSubnetMask*	Pointer to the new subnet mask to be used by the network device.

Return values:

EFI_SUCCESS	The new station IP address and/or subnet mask were updated.
EFI_NOT_STARTED	The PXE Base Code Protocol is in the stopped state.

EFI_INVALID_PARAMETER One or more parameters are invalid.

References _PXEBC_PRIVATE_DATA::Arp, _PXEBC_PRIVATE_DATA::IsAddressOk, _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_PRIVATE_DATA_FROM_PXEBC, PxeBcFlushStationIp(), PxeBcRegisterIp6Address(), _PXEBC_PRIVATE_DATA::StationIp, and _PXEBC_PRIVATE_DATA::SubnetMask.

```
EFI_STATUS EFIAPI EfiPxeBcStart ( IN EFI_PXE_BASE_CODE_PROTOCOL *  This,
                                  IN BOOLEAN                       UseIpv6
                                )
```

Enables the use of the PXE Base Code Protocol functions.

This function enables the use of the PXE Base Code Protocol functions. If the Started field of the EFI_PXE_BASE_CODE_MODE structure is already TRUE, then EFI_ALREADY_STARTED will be returned. If UseIpv6 is TRUE, then IPv6 formatted addresses will be used in this session. If UseIpv6 is FALSE, then IPv4 formatted addresses will be used in this session. If UseIpv6 is TRUE, and the Ipv6Supported field of the EFI_PXE_BASE_CODE_MODE structure is FALSE, then EFI_UNSUPPORTED will be returned. If there is not enough memory or other resources to start the PXE Base Code Protocol, then EFI_OUT_OF_RESOURCES will be returned. Otherwise, the PXE Base Code Protocol will be started.

Parameters:

`[in]` *This* Pointer to the EFI_PXE_BASE_CODE_PROTOCOL instance.

`[in]` *UseIpv6* Specifies the type of IP addresses that are to be used during the session that is being started. Set to TRUE for IPv6, and FALSE for IPv4.

Return values:

EFI_SUCCESS	The PXE Base Code Protocol was started.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
EFI_UNSUPPORTED	UseIpv6 is TRUE, but the Ipv6Supported field of the EFI_PXE_BASE_CODE_MODE structure is FALSE.
EFI_ALREADY_STARTED	The PXE Base Code Protocol is already in the started state.
EFI_INVALID_PARAMETER	The This parameter is NULL or does not point to a valid EFI_PXE_BASE_CODE_PROTOCOL structure.
EFI_OUT_OF_RESOURCES	Could not allocate enough memory or other resources to start the PXE Base Code Protocol.

References PXEBC_DHCP4_PACKET::Ack, PXEBC_DHCP6_PACKET::Ack, _PXEBC_PRIVATE_DATA::ArpUpdateEvent, _PXEBC_PRIVATE_DATA::BlockSize, PXEBC_DHCP_PACKET_CACHE::Dhcp4, PXEBC_DHCP_PACKET_CACHE::Dhcp6, _PXEBC_PRIVATE_DATA::DhcpAck, _PXEBC_PRIVATE_DATA::Icmp6Token, _PXEBC_PRIVATE_DATA::IcmpToken, _PXEBC_PRIVATE_DATA::Ip4, _PXEBC_PRIVATE_DATA::Ip4MaxPacketSize, _PXEBC_PRIVATE_DATA::Ip4Nic, _PXEBC_PRIVATE_DATA::Ip6, _PXEBC_PRIVATE_DATA::Ip6MaxPacketSize, _PXEBC_PRIVATE_DATA::Ip6Nic, _PXEBC_PRIVATE_DATA::Ip6Policy, _PXEBC_PRIVATE_DATA::IsAddressOk, PXEBC_DHCP4_PACKET::Offer, PXEBC_DHCP6_PACKET::Offer, _PXEBC_PRIVATE_DATA::OfferBuffer, PXEBC_DHCP4_PACKET_CACHE::Packet, PXEBC_DHCP6_PACKET_CACHE::Packet, _PXEBC_PRIVATE_DATA::ProxyOffer, _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_DEFAULT_TFTP_OVERHEAD_SIZE, PXEBC_DEFAULT_UDP_OVERHEAD_SIZE, PXEBC_DHCP4_PACKET_MAX_SIZE, PXEBC_DHCP6_PACKET_MAX_SIZE, PXEBC_IP6_POLICY_MAX, PXEBC_OFFER_MAX_NUM, PXEBC_PRIVATE_DATA_FROM_PXEBC, PxeBcArpCacheUpdate(), PxeBcIcmp6ErrorUpdate(), PxeBcIcmpErrorUpdate(), PxeBcSeedDhcp4Packet(),

PxeBcSetIp4Policy(), PxeBcSetIp6Policy(), _PXEBC_PRIVATE_DATA::PxeReply, _PXEBC_PRIVATE_DATA::SeedPacket, _PXEBC_PRIVATE_DATA::Udp4CfgData, _PXEBC_PRIVATE_DATA::Udp4Read, _PXEBC_PRIVATE_DATA::Udp6CfgData, _PXEBC_PRIVATE_DATA::Udp6Read, and _PXEBC_PRIVATE_DATA::UdpTimeOutEvent.

EFI_STATUS EFIAPI EfiPxeBcStop (IN EFI_PXE_BASE_CODE_PROTOCOL * *This*)

Disable the use of the PXE Base Code Protocol functions.

This function stops all activity on the network device. All the resources allocated in Start() are released, the Started field of the EFI_PXE_BASE_CODE_MODE structure is set to FALSE, and EFI_SUCCESS is returned. If the Started field of the EFI_PXE_BASE_CODE_MODE structure is already FALSE, then EFI_NOT_STARTED will be returned.

Parameters:

`[in]` *This* Pointer to the EFI_PXE_BASE_CODE_PROTOCOL instance.

Return values:

EFI_SUCCESS	The PXE Base Code Protocol was stopped.
EFI_NOT_STARTED	The PXE Base Code Protocol is already in the stopped state.
EFI_INVALID_PARAMETER	The This parameter is NULL or does not point to a valid EFI_PXE_BASE_CODE_PROTOCOL structure.
Others	

References _PXEBC_PRIVATE_DATA::ArpUpdateEvent, _PXEBC_PRIVATE_DATA::BootFileName, _PXEBC_PRIVATE_DATA::BootFileSize, _PXEBC_PRIVATE_DATA::CurSrcPort, _PXEBC_PRIVATE_DATA::Dhcp4, _PXEBC_PRIVATE_DATA::Dhcp6, _PXEBC_PRIVATE_DATA::Dhcp6Request, _PXEBC_PRIVATE_DATA::ElapsedTime, _PXEBC_PRIVATE_DATA::GatewayIp, _PXEBC_PRIVATE_DATA::Icmp6Token, _PXEBC_PRIVATE_DATA::IcmpToken, _PXEBC_PRIVATE_DATA::Ip4, _PXEBC_PRIVATE_DATA::Ip4CfgData, _PXEBC_PRIVATE_DATA::Ip6, _PXEBC_PRIVATE_DATA::Ip6CfgData, _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_PRIVATE_DATA_FROM_PXEBC, PxeBcUnregisterIp6Address(), _PXEBC_PRIVATE_DATA::ServerIp, _PXEBC_PRIVATE_DATA::SolicitTimes, _PXEBC_PRIVATE_DATA::StationIp, _PXEBC_PRIVATE_DATA::SubnetMask, _PXEBC_PRIVATE_DATA::Udp4CfgData, _PXEBC_PRIVATE_DATA::Udp4Read, _PXEBC_PRIVATE_DATA::Udp4Write, _PXEBC_PRIVATE_DATA::Udp6CfgData, _PXEBC_PRIVATE_DATA::Udp6Read, _PXEBC_PRIVATE_DATA::Udp6Write, and _PXEBC_PRIVATE_DATA::UdpTimeOutEvent.

EFI_STATUS EFIAPI EfiPxeBcUdpRead (

IN EFI_PXE_BASE_CODE_PROTOCOL *	*This*,
IN UINT16	*OpFlags*,
IN OUT EFI_IP_ADDRESS *	*DestIp*,
IN OUT EFI_PXE_BASE_CODE_UDP_PORT *	*DestPort*,
IN OUT EFI_IP_ADDRESS *	*SrcIp*,
IN OUT EFI_PXE_BASE_CODE_UDP_PORT *	*SrcPort*,
IN UINTN *	*HeaderSize*,
IN VOID *	*HeaderPtr*,

	IN OUT UINTN *	*BufferSize*,
	IN VOID *	*BufferPtr*
)		

Reads a UDP packet from the network interface. + This function reads a UDP packet from a network interface. The data contents are returned in (the optional HeaderPtr and) BufferPtr, and the size of the buffer received is returned in BufferSize . If the input BufferSize is smaller than the UDP packet received (less optional HeaderSize), it will be set to the required size, and EFI_BUFFER_TOO_SMALL will be returned. In this case, the contents of BufferPtr are undefined, and the packet is lost. If a UDP packet is successfully received, then EFI_SUCCESS will be returned, and the information from the UDP header will be returned in DestIp, DestPort, SrcIp, and SrcPort if they are not NULL. Depending on the values of OpFlags and the DestIp, DestPort, SrcIp, and SrcPort input values, different types of UDP packet receive filtering will be performed. The following tables summarize these receive filter operations.

Parameters:

`[in]`	*This*	Pointer to the EFI_PXE_BASE_CODE_PROTOCOL instance.
`[in]`	*OpFlags*	The UDP operation flags.
`[in,out]`	*DestIp*	The destination IP address.
`[in,out]`	*DestPort*	The destination UDP port number.
`[in,out]`	*SrcIp*	The source IP address.
`[in,out]`	*SrcPort*	The source UDP port number.
`[in]`	*HeaderSize*	An optional field which may be set to the length of a header at HeaderPtr to be prefixed to the data at BufferPtr.
`[in]`	*HeaderPtr*	If HeaderSize is not NULL, a pointer to a header to be prefixed to the data at BufferPtr.
`[in,out]`	*BufferSize*	A pointer to the size of the data at BufferPtr.
`[in]`	*BufferPtr*	A pointer to the data to be read.

Return values:

EFI_SUCCESS	The UDP Read operation was completed.
EFI_NOT_STARTED	The PXE Base Code Protocol is in the stopped state.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
EFI_BUFFER_TOO_SMALL	The packet is larger than Buffer can hold.
EFI_ABORTED	The callback function aborted the UDP Read operation.
EFI_TIMEOUT	The UDP Read operation timed out.

References _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_PRIVATE_DATA_FROM_PXEBC, PXEBC_UDP_TIMEOUT, PxeBcCommonNotify(), PxeBcUdp4Read(), PxeBcUdp6Read(), _PXEBC_PRIVATE_DATA::Udp4Read, _PXEBC_PRIVATE_DATA::Udp6Read, and _PXEBC_PRIVATE_DATA::UdpTimeOutEvent.

EFI_STATUS EFIAPI EfiPxeBcUdpWrite	(IN EFI_PXE_BASE_CODE_PROTOCOL *	*This*,
	IN UINT16	*OpFlags*,
	IN EFI_IP_ADDRESS *	*DestIp*,
	IN EFI_PXE_BASE_CODE_UDP_PORT *	*DestPort*,
	IN EFI_IP_ADDRESS *	*GatewayIp*,
	IN EFI_IP_ADDRESS *	*SrcIp*,
	IN OUT EFI_PXE_BASE_CODE_UDP_PORT *	*SrcPort*,

```
                                    IN UINTN *                          HeaderSize,
                                    IN VOID *                           HeaderPtr,
                                    IN UINTN *                          BufferSize,
                                    IN VOID *                           BufferPtr
                                  )
```

Writes a UDP packet to the network interface.

This function writes a UDP packet specified by the (optional HeaderPtr and) BufferPtr parameters to the network interface. The UDP header is automatically built by this routine. It uses the parameters OpFlags, DestIp, DestPort, GatewayIp, SrcIp, and SrcPort to build this header. If the packet is successfully built and transmitted through the network interface, then EFI_SUCCESS will be returned. If a timeout occurs during the transmission of the packet, then EFI_TIMEOUT will be returned. If an ICMP error occurs during the transmission of the packet, then the IcmpErrorReceived field is set to TRUE, the IcmpError field is filled in and EFI_ICMP_ERROR will be returned. If the Callback Protocol does not return EFI_PXE_BASE_CODE_CALLBACK_STATUS_CONTINUE, then EFI_ABORTED will be returned.

Parameters:

`[in]`	*This*	Pointer to the EFI_PXE_BASE_CODE_PROTOCOL instance.
`[in]`	*OpFlags*	The UDP operation flags.
`[in]`	*DestIp*	The destination IP address.
`[in]`	*DestPort*	The destination UDP port number.
`[in]`	*GatewayIp*	The gateway IP address.
`[in]`	*SrcIp*	The source IP address.
`[in,out]`	*SrcPort*	The source UDP port number.
`[in]`	*HeaderSize*	An optional field which may be set to the length of a header at HeaderPtr to be prefixed to the data at BufferPtr.
`[in]`	*HeaderPtr*	If HeaderSize is not NULL, a pointer to a header to be prefixed to the data at BufferPtr.
`[in]`	*BufferSize*	A pointer to the size of the data at BufferPtr.
`[in]`	*BufferPtr*	A pointer to the data to be written.

Return values:

EFI_SUCCESS	The UDP Write operation completed.
EFI_NOT_STARTED	The PXE Base Code Protocol is in the stopped state.
EFI_INVALID_PARAMETER	One or more parameters are invalid.
EFI_BAD_BUFFER_SIZE	The buffer is too long to be transmitted.
EFI_ABORTED	The callback function aborted the UDP Write operation.
EFI_TIMEOUT	The UDP Write operation timed out.
EFI_ICMP_ERROR	An ICMP error packet was received during the UDP write session.

References _PXEBC_PRIVATE_DATA::CurSrcPort, _PXEBC_PRIVATE_DATA::GatewayIp, _PXEBC_PRIVATE_DATA::IsAddressOk, _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_PRIVATE_DATA_FROM_PXEBC, PXEBC_UDP_TIMEOUT, PxeBcConfigUdp4Write(), PxeBcConfigUdp6Write(), PxeBcUdp4Write(), PxeBcUdp6Write(), _PXEBC_PRIVATE_DATA::StationIp, _PXEBC_PRIVATE_DATA::SubnetMask, _PXEBC_PRIVATE_DATA::Udp4Write, _PXEBC_PRIVATE_DATA::Udp6Write, and _PXEBC_PRIVATE_DATA::UdpTimeOutEvent.

```
EFI_STATUS EFIAPI EfiPxeLoadFile  ( IN EFI_LOAD_FILE_PROTOCOL *       This,
                                    IN EFI_DEVICE_PATH_PROTOCOL *     FilePath,
                                    IN BOOLEAN                        BootPolicy,
```

```
IN OUT UINTN *          BufferSize,
IN VOID *               Buffer
)
```

Causes the driver to load a specified file.

Parameters:

[in]	*This*	Protocol instance pointer.
[in]	*FilePath*	The device specific path of the file to load.
[in]	*BootPolicy*	If TRUE, indicates that the request originates from the boot manager is attempting to load FilePath as a boot selection. If FALSE, then FilePath must match an exact file to be loaded.
[in,out]	*BufferSize*	On input the size of Buffer in bytes. On output with a return code of EFI_SUCCESS, the amount of data transferred to Buffer. On output with a return code of EFI_BUFFER_TOO_SMALL, the size of Buffer required to retrieve the requested file.
[in]	*Buffer*	The memory buffer to transfer the file to. IF Buffer is NULL, then no the size of the requested file is returned in BufferSize.

Return values:

EFI_SUCCESS	The file was loaded.
EFI_UNSUPPORTED	The device does not support the provided BootPolicy.
EFI_INVALID_PARAMETER	FilePath is not a valid device path, or BufferSize is NULL.
EFI_NO_MEDIA	No medium was present to load the file.
EFI_DEVICE_ERROR	The file was not loaded due to a device error.
EFI_NO_RESPONSE	The remote system did not respond.
EFI_NOT_FOUND	The file was not found.
EFI_ABORTED	The file load process was manually cancelled.

References _PXEBC_PRIVATE_DATA::Controller, _PXEBC_PRIVATE_DATA::Ip6Nic, _PXEBC_VIRTUAL_NIC::Private, _PXEBC_PRIVATE_DATA::PxeBc, PXEBC_VIRTUAL_NIC_FROM_LOADFILE, and PxeBcLoadBootFile().

```
EFI_PXE_BASE_CODE_CALLBACK_STATUS EFIAPI EfiPxeLoadFileCallback (
  IN EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL *   This,
  IN EFI_PXE_BASE_CODE_FUNCTION              Function,
  IN BOOLEAN                                 Received,
  IN UINT32                                  PacketLength,
  IN EFI_PXE_BASE_CODE_PACKET *              PacketPtr
)
```

Callback function that is invoked when the PXE Base Code Protocol is about to transmit, has received, or is waiting to receive a packet.

This function is invoked when the PXE Base Code Protocol is about to transmit, has received, or is waiting to receive a packet. Parameters Function and Received specify the type of event. Parameters PacketLen and Packet specify the packet that generated the event. If these fields are zero and NULL respectively, then this is a status update callback. If the operation specified by Function is to continue, then CALLBACK_STATUS_CONTINUE should be returned. If the operation specified by Function should be aborted, then CALLBACK_STATUS_ABORT should be returned. Due to the polling nature of UEFI device drivers, a callback function should not execute for more than 5 ms. The SetParameters() function must be

called after a Callback Protocol is installed to enable the use of callbacks.

Parameters:

[in]	*This*	Pointer to the EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL instance.
[in]	*Function*	The PXE Base Code Protocol function that is waiting for an event.
[in]	*Received*	TRUE if the callback is being invoked due to a receive event. FALSE if the callback is being invoked due to a transmit event.
[in]	*PacketLength*	The length, in bytes, of Packet. This field will have a value of zero if this is a wait for receive event.
[in]	*PacketPtr*	If Received is TRUE, a pointer to the packet that was just received; otherwise a pointer to the packet that is about to be transmitted.

Return values:

EFI_PXE_BASE_CODE_CALLBACK_STATUS_CONTINUE	If Function specifies a continue operation.
EFI_PXE_BASE_CODE_CALLBACK_STATUS_ABORT	If Function specifies an abort operation.

Variable Documentation

EFI_LOAD_FILE_PROTOCOL gLoadFileProtocolTemplate = { EfiPxeLoadFile }

Referenced by PxeBcCreateIp4Children(), and PxeBcCreateIp6Children().

EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL gPxeBcCallBackTemplate

Initial value:

```
{
 EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL_REVISION,
 EfiPxeLoadFileCallback
}
```

Referenced by PxeBcInstallCallback().

EFI_PXE_BASE_CODE_PROTOCOL gPxeBcProtocolTemplate

Initial value:

```
{
 EFI_PXE_BASE_CODE_PROTOCOL_REVISION,
 EfiPxeBcStart,
 EfiPxeBcStop,
 EfiPxeBcDhcp,
 EfiPxeBcDiscover,
 EfiPxeBcMtftp,
 EfiPxeBcUdpWrite,
 EfiPxeBcUdpRead,
 EfiPxeBcSetIpFilter,
 EfiPxeBcArp,
 EfiPxeBcSetParameters,
 EfiPxeBcSetStationIP,
 EfiPxeBcSetPackets,
 NULL
}
```

Referenced by PxeBcStart().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/UefiPxeBcDxe/PxeBcImpl.h File Reference

Data Structures

union PXEBC_DHCP_PACKET_CACHE
struct _PXEBC_PRIVATE_PROTOCOL
struct _PXEBC_VIRTUAL_NIC
struct _PXEBC_PRIVATE_DATA

Defines

"memItemRight" valign="bottom">PXEBC_DEFAULT_HOPLIMIT 64
#define PXEBC_DEFAULT_LIFETIME 50000
#define PXEBC_UDP_TIMEOUT 30000000
#define PXEBC_DAD_ADDITIONAL_DELAY 3
#define PXEBC_MTFTP_TIMEOUT 4
#define PXEBC_MTFTP_RETRIES 6
#define PXEBC_DHCP_RETRIES 4
#define PXEBC_MENU_MAX_NUM 24
#define PXEBC_OFFER_MAX_NUM 16
#define PXEBC_PRIVATE_DATA_SIGNATURE
('P', 'X', 'E', 'P')
#define PXEBC_VIRTUAL_NIC_SIGNATURE S
('P', 'X', 'E', 'V')
#define PXEBC_PRIVATE_DATA_FROM_PXEB
(a, PXEBC_PRIVATE_DATA, PxeBc,
PXEBC_PRIVATE_DATA_SIGNATURE)
#define PXEBC_PRIVATE_DATA_FROM_ID(a)
PXEBC_PRIVATE_DATA, Id,
PXEBC_PRIVATE_DATA_SIGNATURE)
#define PXEBC_VIRTUAL_NIC_FROM_LOADF
(a, PXEBC_VIRTUAL_NIC, LoadFile,
PXEBC_VIRTUAL_NIC_SIGNATURE)

Typedefs

typedef struct _PXEBC_PRIVATE_DATA PXEBC_PRIVATE_DATA
typedef struct
_PXEBC_PRIVATE_PROTOCOL PXEBC_PRIVATE_PROTOCOL
typedef struct _PXEBC_VIRTUAL_NIC PXEBC_VIRTUAL_NIC

Variables

EFI_PXE_BASE_CODE_PROTOCOL gPxeBcProtocolTemplate
EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL gPxeBcCallBackTemplate
EFI_LOAD_FILE_PROTOCOL gLoadFileProtocolTemplate

Detailed Description

This EFI_PXE_BASE_CODE_PROTOCOL and EFI_LOAD_FILE_PROTOCOL. interfaces declaration.

Copyright (c) 2007 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by PxeBcDhcp6Sarr(), and PxeBcRegisterIp6Address().

#define PXEBC_DEFAULT_HOPLIMIT 64

Referenced by PxeBcConfigUdp6Write(), and PxeBcCreateIp6Children().

#define PXEBC_DEFAULT_LIFETIME 50000

Referenced by PxeBcConfigUdp4Write(), PxeBcConfigUdp6Write(), PxeBcCreateIp4Children(), and PxeBcCreateIp6Children().

#define PXEBC_DHCP_RETRIES 4

Referenced by PxeBcDhcp4Dora().

#define PXEBC_MENU_MAX_NUM 24

Referenced by PxeBcSelectBootMenu().

#define PXEBC_MTFTP_RETRIES 6

Referenced by EfiPxeBcMtftp().

#define PXEBC_MTFTP_TIMEOUT 4

Referenced by EfiPxeBcMtftp().

#define PXEBC_OFFER_MAX_NUM 16

Referenced by EfiPxeBcStart(), PxeBcCacheDhcp4Offer(), PxeBcCacheDhcp6Offer(), PxeBcCopyDhcp6Proxy(), PxeBcCopyProxyOffer(), PxeBcDhcp4CallBack(), PxeBcDhcp6CallBack(), PxeBcHandleDhcp4Offer(), PxeBcHandleDhcp6Offer(), PxeBcRetryBinlOffer(), and PxeBcRetryDhcp6Binl().

#define PXEBC_PRIVATE_DATA_FROM_ID (a) CR (a, PXEBC_PRIVATE_DATA, Id, PXEBC_PRIVATE_DATA_SIGNATURE)

Referenced by PxeBcStart(), and PxeBcStop().

#define PXEBC_PRIVATE_DATA_FROM_PXEBC (a) CR (a, PXEBC_PRIVATE_DATA, PxeBc, PXEBC_PRIVATE_DATA_SIGNATURE)

Referenced by EfiPxeBcArp(), EfiPxeBcDhcp(), EfiPxeBcDiscover(), EfiPxeBcMtftp(), EfiPxeBcSetIpFilter(), EfiPxeBcSetPackets(), EfiPxeBcSetParameters(), EfiPxeBcSetStationIP(), EfiPxeBcStart(), EfiPxeBcStop(), EfiPxeBcUdpRead(), and EfiPxeBcUdpWrite().

#define PXEBC_PRIVATE_DATA_SIGNATURE SIGNATURE_32 ('P', 'X', 'E', 'P')

Referenced by PxeBcStart().

#define PXEBC_UDP_TIMEOUT 30000000

Referenced by EfiPxeBcUdpRead(), and EfiPxeBcUdpWrite().

#define PXEBC_VIRTUAL_NIC_FROM_LOADFILE (a) CR (a, PXEBC_VIRTUAL_NIC, LoadFile, PXEBC_VIRTUAL_NIC_SIGNATURE)

Referenced by EfiPxeLoadFile(), and PxeBcStop().

#define PXEBC_VIRTUAL_NIC_SIGNATURE SIGNATURE_32 ('P', 'X', 'E', 'V')

Referenced by PxeBcCreateIp4Children(), and PxeBcCreateIp6Children().

Typedef Documentation

typedef struct _PXEBC_PRIVATE_DATA PXEBC_PRIVATE_DATA

typedef struct _PXEBC_PRIVATE_PROTOCOL PXEBC_PRIVATE_PROTOCOL

typedef struct _PXEBC_VIRTUAL_NIC PXEBC_VIRTUAL_NIC

Variable Documentation

EFI_LOAD_FILE_PROTOCOL gLoadFileProtocolTemplate

Referenced by PxeBcCreateIp4Children(), and PxeBcCreateIp6Children().

EFI_PXE_BASE_CODE_CALLBACK_PROTOCOL gPxeBcCallBackTemplate

Referenced by PxeBcInstallCallback().

EFI_PXE_BASE_CODE_PROTOCOL gPxeBcProtocolTemplate

Referenced by PxeBcStart().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/UefiPxeBcDxe/PxeBcMtftp.c File Reference

Functions

EFI_STATUS EFIAPI PxeBcMtftp6CheckPacket (IN EFI_MTFTP6_PROTOCOL *This, IN EFI_MTFTP6_TOKEN *Token, IN UINT16 PacketLen, IN EFI_MTFTP6_PACKET *Packet)

EFI_STATUS PxeBcMtftp6GetFileSize (IN PXEBC_PRIVATE_DATA *Private, IN EFI_MTFTP6_CONFIG_DATA *Config, IN UINT8 *Filename, IN UINTN *BlockSize, IN OUT UINT64 *BufferSize)

EFI_STATUS PxeBcMtftp6ReadFile (IN PXEBC_PRIVATE_DATA *Private, IN EFI_MTFTP6_CONFIG_DATA *Config, IN UINT8 *Filename, IN UINTN *BlockSize, IN UINT8 *BufferPtr, IN OUT UINT64 *BufferSize, IN BOOLEAN DontUseBuffer)

EFI_STATUS PxeBcMtftp6WriteFile (IN PXEBC_PRIVATE_DATA *Private, IN EFI_MTFTP6_CONFIG_DATA *Config, IN UINT8 *Filename, IN BOOLEAN Overwrite, IN UINTN *BlockSize, IN UINT8 *BufferPtr, IN OUT UINT64 *BufferSize)

EFI_STATUS PxeBcMtftp6ReadDirectory (IN PXEBC_PRIVATE_DATA *Private, IN EFI_MTFTP6_CONFIG_DATA *Config, IN UINT8 *Filename, IN UINTN *BlockSize, IN UINT8 *BufferPtr, IN OUT UINT64 *BufferSize, IN BOOLEAN DontUseBuffer)

EFI_STATUS EFIAPI PxeBcMtftp4CheckPacket (IN EFI_MTFTP4_PROTOCOL *This, IN EFI_MTFTP4_TOKEN *Token, IN UINT16 PacketLen, IN EFI_MTFTP4_PACKET *Packet)

EFI_STATUS PxeBcMtftp4GetFileSize (IN PXEBC_PRIVATE_DATA *Private, IN EFI_MTFTP4_CONFIG_DATA *Config, IN UINT8 *Filename, IN UINTN *BlockSize, IN OUT UINT64 *BufferSize)

EFI_STATUS PxeBcMtftp4ReadFile (IN PXEBC_PRIVATE_DATA *Private, IN EFI_MTFTP4_CONFIG_DATA *Config, IN UINT8 *Filename, IN UINTN *BlockSize, IN UINT8 *BufferPtr, IN OUT UINT64 *BufferSize, IN BOOLEAN DontUseBuffer)

EFI_STATUS PxeBcMtftp4WriteFile (IN PXEBC_PRIVATE_DATA *Private, IN EFI_MTFTP4_CONFIG_DATA *Config, IN UINT8 *Filename, IN BOOLEAN Overwrite, IN UINTN *BlockSize, IN UINT8 *BufferPtr, IN OUT UINT64 *BufferSize)

EFI_STATUS PxeBcMtftp4ReadDirectory (IN PXEBC_PRIVATE_DATA *Private, IN EFI_MTFTP4_CONFIG_DATA *Config, IN UINT8 *Filename, IN UINTN *BlockSize, IN UINT8 *BufferPtr, IN OUT UINT64 *BufferSize, IN BOOLEAN DontUseBuffer)

EFI_STATUS PxeBcTftpGetFileSize (IN PXEBC_PRIVATE_DATA *Private, IN VOID *Config, IN UINT8 *Filename, IN UINTN *BlockSize, IN OUT UINT64 *BufferSize)

EFI_STATUS PxeBcTftpReadFile (IN PXEBC_PRIVATE_DATA *Private, IN VOID *Config, IN UINT8 *Filename, IN UINTN *BlockSize, IN UINT8 *BufferPtr, IN OUT UINT64 *BufferSize, IN BOOLEAN DontUseBuffer)

EFI_STATUS PxeBcTftpWriteFile (IN PXEBC_PRIVATE_DATA *Private, IN VOID *Config, IN UINT8 *Filename, IN BOOLEAN Overwrite, IN UINTN *BlockSize, IN UINT8 *BufferPtr, IN OUT UINT64 *BufferSize)

EFI_STATUS PxeBcTftpReadDirectory (IN PXEBC_PRIVATE_DATA *Private, IN VOID *Config, IN UINT8 *Filename, IN UINTN *BlockSize, IN UINT8 *BufferPtr, IN

OUT UINT64 *BufferSize, IN BOOLEAN DontUseBuffer)

Variables

CHAR8 * mMtftpOptions [PXE_MTFTP_OPTION_MAXIMUM_INDEX]

Detailed Description

Functions implementation related with Mtftp for UefiPxeBc Driver.

Copyright (c) 2007 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

```
EFI_STATUS EFIAPI PxeBcMtftp4CheckPacket ( IN EFI_MTFTP4_PROTOCOL *  This,
                                           IN EFI_MTFTP4_TOKEN *    Token,
                                           IN UINT16                PacketLen,
                                           IN EFI_MTFTP4_PACKET *   Packet
                                          )
```

This is a callback function when packets are received or transmitted in Mtftp driver.

A callback function that is provided by the caller to intercept the EFI_MTFTP6_OPCODE_DATA or EFI_MTFTP4_OPCODE_DATA8 packets processed in the EFI_MTFTP4_PROTOCOL.ReadFile() function, and alternatively to intercept EFI_MTFTP4_OPCODE_OACK or EFI_MTFTP4_OPCODE_ERROR packets during a call to EFI_MTFTP4_PROTOCOL.ReadFile(), WriteFile() or ReadDirectory().

Parameters:

[in]	*This*	Pointer to EFI_MTFTP4_PROTOCOL.
[in]	*Token*	Pointer to EFI_MTFTP4_TOKEN.
[in]	*PacketLen*	Length of EFI_MTFTP4_PACKET.
[in]	*Packet*	Pointer to EFI_MTFTP4_PACKET to be checked.

Return values:

EFI_SUCCESS	The current operation succeeeded.
EFI_ABORTED	Abort the current transfer process.

References _PXEBC_PRIVATE_DATA::Function, _PXEBC_PRIVATE_DATA::Mode, PXE_MTFTP_ERROR_STRING_LENGTH, and _PXEBC_PRIVATE_DATA::PxeBcCallback.

Referenced by PxeBcMtftp4ReadDirectory(), PxeBcMtftp4ReadFile(), and PxeBcMtftp4WriteFile().

```
EFI_STATUS PxeBcMtftp4GetFileSize ( IN PXEBC_PRIVATE_DATA *          Private,
                                    IN EFI_MTFTP4_CONFIG_DATA *      Config,
                                    IN UINT8 *                       Filename,
                                    IN UINTN *                       BlockSize,
                                    IN OUT UINT64 *                  BufferSize
                                  )
```

This function is to get size of a file using Tftp.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to EFI_MTFTP4_CONFIG_DATA.
[in]	*Filename*	Pointer to boot file name.
[in]	*BlockSize*	Pointer to required block size.
[in,out]	*BufferSize*	Pointer to buffer size.

Return values:

EFI_SUCCESS	Successfully obtained the size of file.
EFI_NOT_FOUND	Parse the tftp options failed.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
Others	Did not obtain the size of the file.

References mMtftpOptions, PXE_MTFTP_ERROR_STRING_LENGTH, PXE_MTFTP_OPTBUF_MAXNUM_INDEX, PXE_MTFTP_OPTION_BLKSIZE_INDEX, PXE_MTFTP_OPTION_TSIZE_INDEX, PXEBC_BS_DOWNLOAD_PORT, and PxeBcUintnToAscDec().

Referenced by PxeBcTftpGetFileSize().

```
EFI_STATUS PxeBcMtftp4ReadDirectory ( IN PXEBC_PRIVATE_DATA *          Private,
                                      IN EFI_MTFTP4_CONFIG_DATA *      Config,
                                      IN UINT8 *                       Filename,
                                      IN UINTN *                       BlockSize,
                                      IN UINT8 *                       BufferPtr,
                                      IN OUT UINT64 *                  BufferSize,
                                      IN BOOLEAN                       DontUseBuffer
                                    )
```

This function is to get data (file) from a directory using Tftp.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to EFI_MTFTP4_CONFIG_DATA.
[in]	*Filename*	Pointer to boot file name.
[in]	*BlockSize*	Pointer to required block size.
[in]	*BufferPtr*	Pointer to buffer.
[in,out]	*BufferSize*	Pointer to buffer size.
[in]	*DontUseBuffer*	Indicates whether to use a receive buffer.

Return values:

EFI_SUCCES	Successfully obtained the data from the file included in the directory.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.

Others	Operation failed.

References mMtftpOptions, PXE_MTFTP_OPTBUF_MAXNUM_INDEX, PXE_MTFTP_OPTION_BLKSIZE_INDEX, PXEBC_BS_DOWNLOAD_PORT, PxeBcMtftp4CheckPacket(), and PxeBcUintnToAscDec().

Referenced by PxeBcTftpReadDirectory().

```
EFI_STATUS PxeBcMtftp4ReadFile ( IN PXEBC_PRIVATE_DATA *          Private,
                                 IN EFI_MTFTP4_CONFIG_DATA *      Config,
                                 IN UINT8 *                       Filename,
                                 IN UINTN *                       BlockSize,
                                 IN UINT8 *                       BufferPtr,
                                 IN OUT UINT64 *                  BufferSize,
                                 IN BOOLEAN                       DontUseBuffer
                               )
```

This function is to read the data of a file using Tftp.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to EFI_MTFTP4_CONFIG_DATA.
[in]	*Filename*	Pointer to boot file name.
[in]	*BlockSize*	Pointer to required block size.
[in]	*BufferPtr*	Pointer to buffer.
[in,out]	*BufferSize*	Pointer to buffer size.
[in]	*DontUseBuffer*	Indicates whether to use a receive buffer.

Return values:

EFI_SUCCESS	Successfully read the data from the special file.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
Others	Read data from file failed.

References mMtftpOptions, PXE_MTFTP_OPTBUF_MAXNUM_INDEX, PXE_MTFTP_OPTION_BLKSIZE_INDEX, PXEBC_BS_DOWNLOAD_PORT, PxeBcMtftp4CheckPacket(), and PxeBcUintnToAscDec().

Referenced by PxeBcTftpReadFile().

```
EFI_STATUS PxeBcMtftp4WriteFile ( IN PXEBC_PRIVATE_DATA *          Private,
                                  IN EFI_MTFTP4_CONFIG_DATA *      Config,
                                  IN UINT8 *                       Filename,
                                  IN BOOLEAN                       Overwrite,
                                  IN UINTN *                       BlockSize,
                                  IN UINT8 *                       BufferPtr,
                                  IN OUT UINT64 *                  BufferSize
                                )
```

This function is to write the data of a file using Tftp.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to EFI_MTFTP4_CONFIG_DATA.
[in]	*Filename*	Pointer to boot file name.
[in]	*Overwrite*	Indicates whether to use the overwrite attribute.
[in]	*BlockSize*	Pointer to required block size.
[in]	*BufferPtr*	Pointer to buffer.
[in,out]	*BufferSize*	Pointer to buffer size.

Return values:

EFI_SUCCESS	Successfully write the data into the special file.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
other	Write data into file failed.

References mMtftpOptions, PXE_MTFTP_OPTBUF_MAXNUM_INDEX, PXE_MTFTP_OPTION_BLKSIZE_INDEX, PXEBC_BS_DOWNLOAD_PORT, PxeBcMtftp4CheckPacket(), and PxeBcUintnToAscDec().

Referenced by PxeBcTftpWriteFile().

```
EFI_STATUS EFIAPI PxeBcMtftp6CheckPacket ( IN EFI_MTFTP6_PROTOCOL *  This,
                                           IN EFI_MTFTP6_TOKEN *     Token,
                                           IN UINT16                 PacketLen,
                                           IN EFI_MTFTP6_PACKET *    Packet
                                         )
```

This is a callback function when packets are received or transmitted in Mtftp driver.

A callback function that is provided by the caller to intercept the EFI_MTFTP6_OPCODE_DATA or EFI_MTFTP6_OPCODE_DATA8 packets processed in the EFI_MTFTP6_PROTOCOL.ReadFile() function, and alternatively to intercept EFI_MTFTP6_OPCODE_OACK or EFI_MTFTP6_OPCODE_ERROR packets during a call to EFI_MTFTP6_PROTOCOL.ReadFile(), WriteFile() or ReadDirectory().

Parameters:

[in]	*This*	Pointer to EFI_MTFTP6_PROTOCOL.
[in]	*Token*	Pointer to EFI_MTFTP6_TOKEN.
[in]	*PacketLen*	Length of EFI_MTFTP6_PACKET.
[in]	*Packet*	Pointer to EFI_MTFTP6_PACKET to be checked.

Return values:

EFI_SUCCESS	The current operation succeeded.
EFI_ABORTED	Abort the current transfer process.

References _PXEBC_PRIVATE_DATA::Function, _PXEBC_PRIVATE_DATA::Mode, PXE_MTFTP_ERROR_STRING_LENGTH, and _PXEBC_PRIVATE_DATA::PxeBcCallback.

Referenced by PxeBcMtftp6ReadDirectory(), PxeBcMtftp6ReadFile(), and PxeBcMtftp6WriteFile().

```
EFI_STATUS PxeBcMtftp6GetFileSize ( IN PXEBC_PRIVATE_DATA *        Private,
                                    IN EFI_MTFTP6_CONFIG_DATA *    Config,
```

```
                                    IN UINT8 *                    Filename,
                                    IN UINTN *                    BlockSize,
                                    IN OUT UINT64 *               BufferSize
                                  )
```

This function is to get the size of a file using Tftp.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to EFI_MTFTP6_CONFIG_DATA.
[in]	*Filename*	Pointer to boot file name.
[in]	*BlockSize*	Pointer to required block size.
[in,out]	*BufferSize*	Pointer to buffer size.

Return values:

EFI_SUCCESS	Sucessfully obtained the size of file.
EFI_NOT_FOUND	Parse the tftp ptions failed.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
Others	Has not obtained the size of the file.

References mMtftpOptions, PXE_MTFTP_ERROR_STRING_LENGTH, PXE_MTFTP_OPTBUF_MAXNUM_INDEX, PXE_MTFTP_OPTION_BLKSIZE_INDEX, PXE_MTFTP_OPTION_TSIZE_INDEX, PXEBC_BS_DOWNLOAD_PORT, and PxeBcUintnToAscDec().

Referenced by PxeBcTftpGetFileSize().

```
EFI_STATUS PxeBcMtftp6ReadDirectory ( IN PXEBC_PRIVATE_DATA *        Private,
                                      IN EFI_MTFTP6_CONFIG_DATA *    Config,
                                      IN UINT8 *                     Filename,
                                      IN UINTN *                     BlockSize,
                                      IN UINT8 *                     BufferPtr,
                                      IN OUT UINT64 *                BufferSize,
                                      IN BOOLEAN                     DontUseBuffer
                                    )
```

This function is to read the data (file) from a directory using Tftp.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to EFI_MTFTP6_CONFIG_DATA.
[in]	*Filename*	Pointer to boot file name.
[in]	*BlockSize*	Pointer to required block size.
[in]	*BufferPtr*	Pointer to buffer.
[in,out]	*BufferSize*	Pointer to buffer size.
[in]	*DontUseBuffer*	Indicates whether to use a receive buffer.

Return values:

EFI_SUCCESS	Successfully obtained the data from the file included in directory.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
Others	Operation failed.

References mMtftpOptions, PXE_MTFTP_OPTBUF_MAXNUM_INDEX, PXE_MTFTP_OPTION_BLKSIZE_INDEX, PXEBC_BS_DOWNLOAD_PORT, PxeBcMtftp6CheckPacket(), and PxeBcUintnToAscDec().

Referenced by PxeBcTftpReadDirectory().

```
EFI_STATUS PxeBcMtftp6ReadFile ( IN PXEBC_PRIVATE_DATA *          Private,
                                 IN EFI_MTFTP6_CONFIG_DATA *      Config,
                                 IN UINT8 *                       Filename,
                                 IN UINTN *                       BlockSize,
                                 IN UINT8 *                       BufferPtr,
                                 IN OUT UINT64 *                  BufferSize,
                                 IN BOOLEAN                       DontUseBuffer
                               )
```

This function is to get data of a file using Tftp.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to EFI_MTFTP6_CONFIG_DATA.
[in]	*Filename*	Pointer to boot file name.
[in]	*BlockSize*	Pointer to required block size.
[in]	*BufferPtr*	Pointer to buffer.
[in,out]	*BufferSize*	Pointer to buffer size.
[in]	*DontUseBuffer*	Indicates whether with a receive buffer.

Return values:

EFI_SUCCESS	Successfully read the data from the special file.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
Others	Read data from file failed.

References mMtftpOptions, PXE_MTFTP_OPTBUF_MAXNUM_INDEX, PXE_MTFTP_OPTION_BLKSIZE_INDEX, PXEBC_BS_DOWNLOAD_PORT, PxeBcMtftp6CheckPacket(), and PxeBcUintnToAscDec().

Referenced by PxeBcTftpReadFile().

```
EFI_STATUS PxeBcMtftp6WriteFile ( IN PXEBC_PRIVATE_DATA *          Private,
                                  IN EFI_MTFTP6_CONFIG_DATA *      Config,
                                  IN UINT8 *                       Filename,
                                  IN BOOLEAN                       Overwrite,
                                  IN UINTN *                       BlockSize,
                                  IN UINT8 *                       BufferPtr,
                                  IN OUT UINT64 *                  BufferSize
                                )
```

This function is used to write the data of a file using Tftp.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.

[in]	*Config*	Pointer to EFI_MTFTP6_CONFIG_DATA.
[in]	*Filename*	Pointer to boot file name.
[in]	*Overwrite*	Indicate whether with overwrite attribute.
[in]	*BlockSize*	Pointer to required block size.
[in]	*BufferPtr*	Pointer to buffer.
[in,out]	*BufferSize*	Pointer to buffer size.

Return values:

EFI_SUCCESS	Successfully wrote the data into a special file.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
other	Write data into file failed.

References mMtftpOptions, PXE_MTFTP_OPTBUF_MAXNUM_INDEX, PXE_MTFTP_OPTION_BLKSIZE_INDEX, PXEBC_BS_DOWNLOAD_PORT, PxeBcMtftp6CheckPacket(), and PxeBcUintnToAscDec().

Referenced by PxeBcTftpWriteFile().

```
EFI_STATUS PxeBcTftpGetFileSize ( IN PXEBC_PRIVATE_DATA *   Private,
                                  IN VOID *                 Config,
                                  IN UINT8 *                Filename,
                                  IN UINTN *                BlockSize,
                                  IN OUT UINT64 *           BufferSize
                                )
```

This function is wrapper to get the file size using TFTP.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to configure data.
[in]	*Filename*	Pointer to boot file name.
[in]	*BlockSize*	Pointer to required block size.
[in,out]	*BufferSize*	Pointer to buffer size.

Return values:

EFI_SUCCESS	Successfully obtained the size of file.
EFI_NOT_FOUND	Parse the tftp options failed.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
Others	Did not obtain the size of the file.

References PxeBcMtftp4GetFileSize(), and PxeBcMtftp6GetFileSize().

Referenced by EfiPxeBcMtftp().

```
EFI_STATUS PxeBcTftpReadDirectory ( IN PXEBC_PRIVATE_DATA *   Private,
                                    IN VOID *                 Config,
                                    IN UINT8 *                Filename,
                                    IN UINTN *                BlockSize,
                                    IN UINT8 *                BufferPtr,
```

```
                                    IN OUT UINT64 *          BufferSize,
                                    IN BOOLEAN               DontUseBuffer
                                  )
```

This function is a wrapper to get the data (file) from a directory using TFTP.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to config data.
[in]	*Filename*	Pointer to boot file name.
[in]	*BlockSize*	Pointer to required block size.
[in]	*BufferPtr*	Pointer to buffer.
[in,out]	*BufferSize*	Pointer to buffer size.
[in]	*DontUseBuffer*	Indicatse whether to use a receive buffer.

Return values:

EFI_SUCCES	Successfully obtained the data from the file included in the directory.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
Others	Operation failed.

References PxeBcMtftp4ReadDirectory(), and PxeBcMtftp6ReadDirectory().

Referenced by EfiPxeBcMtftp().

```
EFI_STATUS PxeBcTftpReadFile ( IN PXEBC_PRIVATE_DATA *  Private,
                               IN VOID *                Config,
                               IN UINT8 *               Filename,
                               IN UINTN *               BlockSize,
                               IN UINT8 *               BufferPtr,
                               IN OUT UINT64 *          BufferSize,
                               IN BOOLEAN               DontUseBuffer
                             )
```

This function is a wrapper to get file using TFTP.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to config data.
[in]	*Filename*	Pointer to boot file name.
[in]	*BlockSize*	Pointer to required block size.
[in]	*BufferPtr*	Pointer to buffer.
[in,out]	*BufferSize*	Pointer to buffer size.
[in]	*DontUseBuffer*	Indicates whether to use a receive buffer.

Return values:

EFI_SUCCESS	Sucessfully read the data from the special file.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
Others	Read data from file failed.

References PxeBcMtftp4ReadFile(), and PxeBcMtftp6ReadFile().

Referenced by EfiPxeBcMtftp().

```
EFI_STATUS PxeBcTftpWriteFile ( IN PXEBC_PRIVATE_DATA *   Private,
                                IN VOID *                 Config,
                                IN UINT8 *                Filename,
                                IN BOOLEAN                Overwrite,
                                IN UINTN *                BlockSize,
                                IN UINT8 *                BufferPtr,
                                IN OUT UINT64 *           BufferSize
                              )
```

This function is a wrapper to write file using TFTP.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to config data.
[in]	*Filename*	Pointer to boot file name.
[in]	*Overwrite*	Indicate whether with overwrite attribute.
[in]	*BlockSize*	Pointer to required block size.
[in]	*BufferPtr*	Pointer to buffer.
[in,out]	*BufferSize*	Pointer to buffer size.

Return values:

EFI_SUCCESS	Successfully wrote the data into a special file.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
other	Write data into file failed.

References PxeBcMtftp4WriteFile(), and PxeBcMtftp6WriteFile().

Referenced by EfiPxeBcMtftp().

Variable Documentation

CHAR8* mMtftpOptions[PXE_MTFTP_OPTION_MAXIMUM_INDEX]

Initial value:

```
{
 "blksize",
 "timeout",
 "tsize",
 "multicast"
}
```

Referenced by PxeBcMtftp4GetFileSize(), PxeBcMtftp4ReadDirectory(), PxeBcMtftp4ReadFile(), PxeBcMtftp4WriteFile(), PxeBcMtftp6GetFileSize(), PxeBcMtftp6ReadDirectory(), PxeBcMtftp6ReadFile(), and PxeBcMtftp6WriteFile().

- File List
- Globals

NetworkPkg/UefiPxeBcDxe/PxeBcMtftp.h File Reference

Defines

"memItemRight" valign="bottom">PXE_MTFTP_OPTION_BLKSIZE_INDEX 0

#define PXE_MTFTP_OPTION_TIMEO
#define PXE_MTFTP_OPTION_TSIZE_
#define PXE_MTFTP_OPTION_MULTI
#define PXE_MTFTP_OPTION_MAXIM
#define PXE_MTFTP_OPTBUF_MAXN
#define PXE_MTFTP_ERROR_STRING
#define PXE_MTFTP_DEFAULT_BLO

Functions

EFI_STATUS PxeBcTftpGetFileSize (IN PXEBC_PRIVATE_DATA *Priv *Config, IN UINT8 *Filename, I *BlockSize, IN OUT UINT64 *B

EFI_STATUS PxeBcTftpReadFile (IN PXEBC_ *Private, IN VOID *Config, IN U IN UINTN *BlockSize, IN UINT OUT UINT64 *BufferSize, IN B DontUseBuffer)

EFI_STATUS PxeBcTftpWriteFile (IN PXEBC_PRIVATE_DATA *Priv *Config, IN UINT8 *Filename, I Overwrite, IN UINTN *BlockSiz *BufferPtr, IN OUT UINT64 *Bu

EFI_STATUS PxeBcTftpReadDirectory (IN PXEBC_PRIVATE_DATA *Priv *Config, IN UINT8 *Filename, I *BlockSize, IN UINT8 *BufferPt UINT64 *BufferSize, IN BOOLE DontUseBuffer)

Detailed Description

Functions declaration related with Mtftp for UefiPxeBc Driver.

Copyright (c) 2007 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by EfiPxeBcMtftp().

#define PXE_MTFTP_ERROR_STRING_LENGTH 127

Referenced by PxeBcMtftp4CheckPacket(), PxeBcMtftp4GetFileSize(), PxeBcMtftp6CheckPacket(), and PxeBcMtftp6GetFileSize().

#define PXE_MTFTP_OPTBUF_MAXNUM_INDEX 128

Referenced by PxeBcMtftp4GetFileSize(), PxeBcMtftp4ReadDirectory(), PxeBcMtftp4ReadFile(), PxeBcMtftp4WriteFile(), PxeBcMtftp6GetFileSize(), PxeBcMtftp6ReadDirectory(), PxeBcMtftp6ReadFile(), and PxeBcMtftp6WriteFile().

#define PXE_MTFTP_OPTION_BLKSIZE_INDEX 0

Referenced by PxeBcMtftp4GetFileSize(), PxeBcMtftp4ReadDirectory(), PxeBcMtftp4ReadFile(), PxeBcMtftp4WriteFile(), PxeBcMtftp6GetFileSize(), PxeBcMtftp6ReadDirectory(), PxeBcMtftp6ReadFile(), and PxeBcMtftp6WriteFile().

#define PXE_MTFTP_OPTION_MAXIMUM_INDEX 4

#define PXE_MTFTP_OPTION_MULTICAST_INDEX 3

#define PXE_MTFTP_OPTION_TIMEOUT_INDEX 1

#define PXE_MTFTP_OPTION_TSIZE_INDEX 2

Referenced by PxeBcMtftp4GetFileSize(), and PxeBcMtftp6GetFileSize().

Function Documentation

```
EFI_STATUS PxeBcTftpGetFileSize ( IN PXEBC_PRIVATE_DATA *   Private,
                                  IN VOID *                 Config,
                                  IN UINT8 *                Filename,
                                  IN UINTN *                BlockSize,
                                  IN OUT UINT64 *           BufferSize
                                )
```

This function is wrapper to get the file size using TFTP.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to configure data.
[in]	*Filename*	Pointer to boot file name.
[in]	*BlockSize*	Pointer to required block size.
[in,out]	*BufferSize*	Pointer to buffer size.

Return values:

EFI_SUCCESS	Successfully obtained the size of file.
EFI_NOT_FOUND	Parse the tftp ptions failed.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
Others	Did not obtain the size of the file.

This function is wrapper to get the file size using TFTP.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to configure data.
[in]	*Filename*	Pointer to boot file name.
[in]	*BlockSize*	Pointer to required block size.
[in,out]	*BufferSize*	Pointer to buffer size.

Return values:

EFI_SUCCESS	Successfully obtained the size of file.
EFI_NOT_FOUND	Parse the tftp options failed.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
Others	Did not obtain the size of the file.

References PxeBcMtftp4GetFileSize(), and PxeBcMtftp6GetFileSize().

Referenced by EfiPxeBcMtftp().

```
EFI_STATUS PxeBcTftpReadDirectory ( IN PXEBC_PRIVATE_DATA *   Private,
                                    IN VOID *                 Config,
                                    IN UINT8 *                Filename,
                                    IN UINTN *                BlockSize,
                                    IN UINT8 *                BufferPtr,
                                    IN OUT UINT64 *           BufferSize,
                                    IN BOOLEAN                DontUseBuffer
                                  )
```

This function is a wrapper to get the data (file) from a directory using TFTP.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to config data.
[in]	*Filename*	Pointer to boot file name.
[in]	*BlockSize*	Pointer to required block size.
[in]	*BufferPtr*	Pointer to buffer.
[in,out]	*BufferSize*	Pointer to buffer size.
[in]	*DontUseBuffer*	Indicates whether with a receive buffer.

Return values:

EFI_SUCCES	Successfully obtained the data from the file included in directory.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
Others	Operation failed.

This function is a wrapper to get the data (file) from a directory using TFTP.

Parameters:

`[in]`	*Private*	Pointer to PxeBc private data.
`[in]`	*Config*	Pointer to config data.
`[in]`	*Filename*	Pointer to boot file name.
`[in]`	*BlockSize*	Pointer to required block size.
`[in]`	*BufferPtr*	Pointer to buffer.
`[in,out]`	*BufferSize*	Pointer to buffer size.
`[in]`	*DontUseBuffer*	Indicatse whether to use a receive buffer.

Return values:

EFI_SUCCES	Successfully obtained the data from the file included in the directory.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
Others	Operation failed.

References PxeBcMtftp4ReadDirectory(), and PxeBcMtftp6ReadDirectory().

Referenced by EfiPxeBcMtftp().

```
EFI_STATUS PxeBcTftpReadFile ( IN PXEBC_PRIVATE_DATA *   Private,
                               IN VOID *                 Config,
                               IN UINT8 *                Filename,
                               IN UINTN *                BlockSize,
                               IN UINT8 *                BufferPtr,
                               IN OUT UINT64 *           BufferSize,
                               IN BOOLEAN                DontUseBuffer
                             )
```

This function is a wrapper to get a file using TFTP.

Parameters:

`[in]`	*Private*	Pointer to PxeBc private data.
`[in]`	*Config*	Pointer to config data.
`[in]`	*Filename*	Pointer to boot file name.
`[in]`	*BlockSize*	Pointer to required block size.
`[in]`	*BufferPtr*	Pointer to buffer.
`[in,out]`	*BufferSize*	Pointer to buffer size.
`[in]`	*DontUseBuffer*	Indicates whether to use a receive buffer.

Return values:

EFI_SUCCESS	Successfully read the data from the special file.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
Others	Read data from file failed.

This function is a wrapper to get file using TFTP.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to config data.
[in]	*Filename*	Pointer to boot file name.
[in]	*BlockSize*	Pointer to required block size.
[in]	*BufferPtr*	Pointer to buffer.
[in,out]	*BufferSize*	Pointer to buffer size.
[in]	*DontUseBuffer*	Indicates whether to use a receive buffer.

Return values:

EFI_SUCCESS	Sucessfully read the data from the special file.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
Others	Read data from file failed.

References PxeBcMtftp4ReadFile(), and PxeBcMtftp6ReadFile().

Referenced by EfiPxeBcMtftp().

```
EFI_STATUS PxeBcTftpWriteFile ( IN PXEBC_PRIVATE_DATA *   Private,
                                IN VOID *                 Config,
                                IN UINT8 *                Filename,
                                IN BOOLEAN                Overwrite,
                                IN UINTN *                BlockSize,
                                IN UINT8 *                BufferPtr,
                                IN OUT UINT64 *           BufferSize
                              )
```

This function is a wrapper to put file with TFTP.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to config data.
[in]	*Filename*	Pointer to boot file name.
[in]	*Overwrite*	Indicates whether to use an overwrite attribute.
[in]	*BlockSize*	Pointer to required block size.
[in]	*BufferPtr*	Pointer to buffer.
[in,out]	*BufferSize*	Pointer to buffer size.

Return values:

EFI_SUCCESS	Successfully wrote the data into the special file.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
other	Write data into file failed.

This function is a wrapper to write file using TFTP.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*Config*	Pointer to config data.
[in]	*Filename*	Pointer to boot file name.
[in]	*Overwrite*	Indicate whether with overwrite attribute.
[in]	*BlockSize*	Pointer to required block size.

[in]	*BufferPtr*	Pointer to buffer.
[in,out]	*BufferSize*	Pointer to buffer size.

Return values:

EFI_SUCCESS	Successfully wrote the data into a special file.
EFI_DEVICE_ERROR	The network device encountered an error during this operation.
other	Write data into file failed.

References PxeBcMtftp4WriteFile(), and PxeBcMtftp6WriteFile().

Referenced by EfiPxeBcMtftp().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

NetworkPkg/UefiPxeBcDxe/PxeBcSupport.c File Reference

Functions

EFI_STATUS PxeBcFlushStationIp (PXEBC_PRIVATE_DATA *Private, EFI_IP_ADDRESS *StationIp, EFI_IP_ADDRESS *SubnetMask)
VOID EFIAPI PxeBcCommonNotify (IN EFI_EVENT Event, IN VOID *Context)
BOOLEAN PxeBcCheckArpCache (IN EFI_PXE_BASE_CODE_MODE *Mode, IN EFI_IPv4_ADDRESS *Ip4Addr, OUT EFI_MAC_ADDRESS *MacAddress)
VOID EFIAPI PxeBcArpCacheUpdate (IN EFI_EVENT Event, IN VOID *Context)
VOID EFIAPI PxeBcIcmpErrorDpcHandle (IN VOID *Context)
VOID EFIAPI PxeBcIcmpErrorUpdate (IN EFI_EVENT Event, IN VOID *Context)
VOID EFIAPI PxeBcIcmp6ErrorDpcHandle (IN VOID *Context)
VOID EFIAPI PxeBcIcmp6ErrorUpdate (IN EFI_EVENT Event, IN VOID *Context)
EFI_STATUS PxeBcConfigUdp4Write (IN EFI_UDP4_PROTOCOL *Udp4, IN EFI_IPv4_ADDRESS *StationIp, IN EFI_IPv4_ADDRESS *SubnetMask, IN EFI_IPv4_ADDRESS *Gateway, IN OUT UINT16 *SrcPort, IN BOOLEAN DoNotFragment)
EFI_STATUS PxeBcConfigUdp6Write (IN EFI_UDP6_PROTOCOL *Udp6, IN EFI_IPv6_ADDRESS *StationIp, IN OUT UINT16 *SrcPort)
EFI_STATUS PxeBcUdp4Write (IN EFI_UDP4_PROTOCOL *Udp4, IN EFI_UDP4_SESSION_DATA *Session, IN EFI_EVENT TimeoutEvent, IN EFI_IPv4_ADDRESS *Gateway, IN UINTN *HeaderSize, IN VOID *HeaderPtr, IN UINTN *BufferSize, IN VOID *BufferPtr)
EFI_STATUS PxeBcUdp6Write (IN EFI_UDP6_PROTOCOL *Udp6, IN EFI_UDP6_SESSION_DATA *Session, IN EFI_EVENT TimeoutEvent, IN UINTN *HeaderSize, IN VOID *HeaderPtr, IN UINTN *BufferSize, IN VOID *BufferPtr)
BOOLEAN PxeBcCheckByIpFilter (IN EFI_PXE_BASE_CODE_MODE *Mode, IN VOID *Session, IN UINT16 OpFlags)
BOOLEAN PxeBcCheckByDestIp (IN EFI_PXE_BASE_CODE_MODE *Mode, IN VOID *Session, IN OUT EFI_IP_ADDRESS *DestIp, IN UINT16 OpFlags)
BOOLEAN PxeBcCheckByDestPort (IN EFI_PXE_BASE_CODE_MODE *Mode, IN VOID *Session, IN OUT UINT16 *DestPort, IN UINT16 OpFlags)
BOOLEAN PxeBcFilterBySrcIp (IN EFI_PXE_BASE_CODE_MODE *Mode, IN VOID *Session, IN OUT EFI_IP_ADDRESS *SrcIp, IN UINT16 OpFlags)
BOOLEAN PxeBcFilterBySrcPort (IN EFI_PXE_BASE_CODE_MODE *Mode, IN VOID *Session, IN OUT UINT16 *SrcPort, IN UINT16 OpFlags)
EFI_STATUS PxeBcUdp4Read (IN EFI_UDP4_PROTOCOL *Udp4, IN EFI_UDP4_COMPLETION_TOKEN *Token, IN EFI_PXE_BASE_CODE_MODE *Mode, IN EFI_EVENT TimeoutEvent, IN UINT16 OpFlags, IN BOOLEAN *IsDone, OUT BOOLEAN *IsMatched, IN OUT EFI_IP_ADDRESS *DestIp, IN OUT EFI_PXE_BASE_CODE_UDP_PORT *DestPort, IN OUT EFI_IP_ADDRESS *SrcIp, IN OUT EFI_PXE_BASE_CODE_UDP_PORT *SrcPort)
EFI_STATUS PxeBcUdp6Read (IN EFI_UDP6_PROTOCOL *Udp6, IN EFI_UDP6_COMPLETION_TOKEN *Token, IN EFI_PXE_BASE_CODE_MODE *Mode, IN EFI_EVENT TimeoutEvent, IN UINT16 OpFlags, IN BOOLEAN *IsDone, OUT BOOLEAN *IsMatched, IN OUT EFI_IP_ADDRESS *DestIp, IN OUT EFI_PXE_BASE_CODE_UDP_PORT *DestPort, IN OUT EFI_IP_ADDRESS *SrcIp, IN OUT EFI_PXE_BASE_CODE_UDP_PORT *SrcPort)
VOID PxeBcShowIp4Addr (IN EFI_IPv4_ADDRESS *Ip)
VOID PxeBcShowIp6Addr (IN EFI_IPv6_ADDRESS *Ip)
VOID

PxeBcUintnToAscDecWithFormat (IN UINTN Number, IN UINT8 *Buffer, IN INTN Length)
UINTN PxeBcUintnToAscDec (IN UINTN Number, IN UINT8 *Buffer, IN UINTN BufferSize)
EFI_STATUS PxeBcUniHexToUint8 (OUT UINT8 *Digit, IN CHAR16 Char)
VOID CalcElapsedTime (IN PXEBC_PRIVATE_DATA *Private)

Detailed Description

Support functions implementation for UefiPxeBc Driver.

Copyright (c) 2007 - 2015, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Function Documentation

VOID CalcElapsedTime (IN PXEBC_PRIVATE_DATA * *Private*)

Calculate the elapsed time.

Parameters:

`[in]` *Private* The pointer to PXE private data

Referenced by PxeBcDhcp6CallBack(), and PxeBcRequestBootService().

VOID EFIAPI PxeBcArpCacheUpdate (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

Update the arp cache periodically.

Parameters:

Event The pointer to EFI_PXE_BC_PROTOCOL.
Context Context of the timer event.

References _PXEBC_PRIVATE_DATA::Arp, and _PXEBC_PRIVATE_DATA::PxeBc.

Referenced by EfiPxeBcArp(), and EfiPxeBcStart().

BOOLEAN PxeBcCheckArpCache (IN EFI_PXE_BASE_CODE_MODE * *Mode*,
IN EFI_IPv4_ADDRESS * *Ip4Addr*,
OUT EFI_MAC_ADDRESS * *MacAddress*
)

Do arp resolution from arp cache in PxeBcMode.

Parameters:

Mode	The pointer to EFI_PXE_BASE_CODE_MODE.
Ip4Addr	The Ip4 address for resolution.
MacAddress	The resoluted MAC address if the resolution is successful. The value is undefined if the resolution fails.

Return values:

TRUE	Found an matched entry.
FALSE	Did not find a matched entry.

Referenced by EfiPxeBcArp().

```
BOOLEAN PxeBcCheckByDestIp ( IN EFI_PXE_BASE_CODE_MODE *  Mode,
                             IN VOID *                    Session,
                             IN OUT EFI_IP_ADDRESS *      DestIp,
                             IN UINT16                    OpFlags
                           )
```

Filter the received packet using the destination Ip.

Parameters:

[in]	*Mode*	The pointer to the mode data of PxeBc.
[in]	*Session*	The pointer to the current UDPv4 session.
[in,out]	*DestIp*	The pointer to the destination Ip address.
[in]	*OpFlags*	Operation flag for UdpRead/UdpWrite.

Return values:

TRUE	Passed the IPv4 filter successfully.
FALSE	Failed to pass the IPv4 filter.

Referenced by PxeBcUdp4Read(), and PxeBcUdp6Read().

```
BOOLEAN PxeBcCheckByDestPort ( IN EFI_PXE_BASE_CODE_MODE *  Mode,
                               IN VOID *                    Session,
                               IN OUT UINT16 *              DestPort,
                               IN UINT16                    OpFlags
                             )
```

Check the received packet using the destination port.

Parameters:

[in]	*Mode*	The pointer to the mode data of PxeBc.
[in]	*Session*	The pointer to the current UDPv4 session.
[in,out]	*DestPort*	The pointer to the destination port.
[in]	*OpFlags*	Operation flag for UdpRead/UdpWrite.

Return values:

TRUE	Passed the IPv4 filter successfully.
FALSE	Failed to pass the IPv4 filter.

Referenced by PxeBcUdp4Read(), and PxeBcUdp6Read().

```
BOOLEAN PxeBcCheckByIpFilter ( IN EFI_PXE_BASE_CODE_MODE *  Mode,
                               IN VOID *                    Session,
                               IN UINT16                    OpFlags
                             )
```

Check the received packet using the Ip filter.

Parameters:

[in]	*Mode*	The pointer to the mode data of PxeBc.
[in]	*Session*	The pointer to the current UDPv4 session.
[in]	*OpFlags*	Operation flag for UdpRead/UdpWrite.

Return values:

TRUE	Passed the Ip filter successfully.
FALSE	Failed to pass the Ip filter.

Referenced by PxeBcUdp4Read(), and PxeBcUdp6Read().

```
VOID EFIAPI PxeBcCommonNotify ( IN EFI_EVENT  Event,
                                IN VOID *     Context
                              )
```

Notify the callback function when an event is triggered.

Parameters:

[in]	*Event*	The triggered event.
[in]	*Context*	The opaque parameter to the function.

Referenced by EfiPxeBcArp(), EfiPxeBcUdpRead(), PxeBcRegisterIp6Address(), PxeBcUdp4Write(), and PxeBcUdp6Write().

```
EFI_STATUS PxeBcConfigUdp4Write ( IN EFI_UDP4_PROTOCOL *  Udp4,
                                  IN EFI_IPv4_ADDRESS *   StationIp,
                                  IN EFI_IPv4_ADDRESS *   SubnetMask,
                                  IN EFI_IPv4_ADDRESS *   Gateway,
                                  IN OUT UINT16 *         SrcPort,
                                  IN BOOLEAN              DoNotFragment
                                )
```

This function is to configure a UDPv4 instance for UdpWrite.

Parameters:

[in]	*Udp4*	The pointer to EFI_UDP4_PROTOCOL.
[in]	*StationIp*	The pointer to the station address.
[in]	*SubnetMask*	The pointer to the subnet mask.
[in]	*Gateway*	The pointer to the gateway address.
[in,out]	*SrcPort*	The pointer to the source port.
[in]	*DoNotFragment*	If TRUE, fragment is not enabled. Otherwise, fragment is enabled.

Return values:

EFI_SUCCESS	Successfully configured this instance.

Others Failed to configure this instance.

References PXEBC_DEFAULT_LIFETIME.

Referenced by EfiPxeBcUdpWrite().

```
EFI_STATUS PxeBcConfigUdp6Write ( IN EFI_UDP6_PROTOCOL *   Udp6,
                                  IN EFI_IPv6_ADDRESS *    StationIp,
                                  IN OUT UINT16 *          SrcPort
                                )
```

This function is to configure a UDPv6 instance for UdpWrite.

Parameters:

[in]	*Udp6*	The pointer to EFI_UDP6_PROTOCOL.
[in]	*StationIp*	The pointer to the station address.
[in,out]	*SrcPort*	The pointer to the source port.

Return values:

EFI_SUCCESS	Successfully configured this instance.
Others	Failed to configure this instance.

References PXEBC_DEFAULT_HOPLIMIT, and PXEBC_DEFAULT_LIFETIME.

Referenced by EfiPxeBcUdpWrite().

```
BOOLEAN PxeBcFilterBySrcIp ( IN EFI_PXE_BASE_CODE_MODE *   Mode,
                             IN VOID *                     Session,
                             IN OUT EFI_IP_ADDRESS *       SrcIp,
                             IN UINT16                     OpFlags
                           )
```

Filter the received packet using the source Ip.

Parameters:

[in]	*Mode*	The pointer to the mode data of PxeBc.
[in]	*Session*	The pointer to the current UDPv4 session.
[in,out]	*SrcIp*	The pointer to the source Ip address.
[in]	*OpFlags*	Operation flag for UdpRead/UdpWrite.

Return values:

TRUE	Passed the IPv4 filter successfully.
FALSE	Failed to pass the IPv4 filter.

Referenced by PxeBcUdp4Read(), and PxeBcUdp6Read().

```
BOOLEAN PxeBcFilterBySrcPort ( IN EFI_PXE_BASE_CODE_MODE *   Mode,
                               IN VOID *                     Session,
                               IN OUT UINT16 *               SrcPort,
                               IN UINT16                     OpFlags
```

)

Filter the received packet using the source port.

Parameters:

[in]	*Mode*	The pointer to the mode data of PxeBc.
[in]	*Session*	The pointer to the current UDPv4 session.
[in,out]	*SrcPort*	The pointer to the source port.
[in]	*OpFlags*	Operation flag for UdpRead/UdpWrite.

Return values:

TRUE	Passed the IPv4 filter successfully.
FALSE	Failed to pass the IPv4 filter.

Referenced by PxeBcUdp4Read(), and PxeBcUdp6Read().

EFI_STATUS PxeBcFlushStationIp (PXEBC_PRIVATE_DATA * *Private*,
EFI_IP_ADDRESS * *StationIp*,
EFI_IP_ADDRESS * *SubnetMask*
)

Flush the previous configration using the new station Ip address.

Parameters:

[in]	*Private*	The pointer to the PxeBc private data.
[in]	*StationIp*	The pointer to the station Ip address.
[in]	*SubnetMask*	The pointer to the subnet mask address for v4.

Return values:

EFI_SUCCESS	Successfully flushed the previous configuration.
Others	Failed to flush using the new station Ip.

References _PXEBC_PRIVATE_DATA::Icmp6Token, _PXEBC_PRIVATE_DATA::IcmpToken, _PXEBC_PRIVATE_DATA::Ip4, _PXEBC_PRIVATE_DATA::Ip4CfgData, _PXEBC_PRIVATE_DATA::Ip6, _PXEBC_PRIVATE_DATA::Ip6CfgData, _PXEBC_PRIVATE_DATA::PxeBc, _PXEBC_PRIVATE_DATA::Udp4CfgData, and _PXEBC_PRIVATE_DATA::Udp6CfgData.

Referenced by EfiPxeBcSetStationIP(), PxeBcDhcp4Dora(), and PxeBcSetIp6Address().

VOID EFIAPI PxeBcIcmp6ErrorDpcHandle (IN VOID * *Context*)

Notify function to handle the received ICMP6 message in DPC.

Parameters:

Context The PXEBC private data.

References _PXEBC_PRIVATE_DATA::Icmp6Token, _PXEBC_PRIVATE_DATA::Ip6, and _PXEBC_PRIVATE_DATA::Mode.

Referenced by PxeBcIcmp6ErrorUpdate().

VOID EFIAPI PxeBcIcmp6ErrorUpdate (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

Callback function to update the latest ICMP6 error message.

Parameters:

Event The event signalled.
Context The context passed in using the event notifier.

References PxeBcIcmp6ErrorDpcHandle().

Referenced by EfiPxeBcStart().

VOID EFIAPI PxeBcIcmpErrorDpcHandle (IN VOID * *Context*)

Notify function to handle the received ICMP message in DPC.

Parameters:

Context The PXEBC private data.

References ICMP_DEST_UNREACHABLE, ICMP_PARAMETER_PROBLEM, ICMP_REDIRECT, ICMP_SOURCE_QUENCH, ICMP_TIME_EXCEEDED, _PXEBC_PRIVATE_DATA::IcmpToken, _PXEBC_PRIVATE_DATA::Ip4, and _PXEBC_PRIVATE_DATA::Mode.

Referenced by PxeBcIcmpErrorUpdate().

VOID EFIAPI PxeBcIcmpErrorUpdate (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

Callback function to update the latest ICMP6 error message.

Parameters:

Event The event signalled.
Context The context passed in using the event notifier.

References PxeBcIcmpErrorDpcHandle().

Referenced by EfiPxeBcStart().

VOID PxeBcShowIp4Addr (IN EFI_IPv4_ADDRESS * *Ip*)

This function is to display the IPv4 address.

Parameters:

`[in]` *Ip* The pointer to the IPv4 address.

Referenced by PxeBcDhcp4BootInfo(), and PxeBcDhcp4Dora().

VOID PxeBcShowIp6Addr (IN EFI_IPv6_ADDRESS * *Ip*)
This function is to display the IPv6 address.

Parameters:

[in] *Ip* The pointer to the IPv6 address.

Referenced by PxeBcDhcp6BootInfo(), and PxeBcSetIp6Address().

```
EFI_STATUS PxeBcUdp4Read ( IN EFI_UDP4_PROTOCOL *                    Udp4,
                           IN EFI_UDP4_COMPLETION_TOKEN *            Token,
                           IN EFI_PXE_BASE_CODE_MODE *               Mode,
                           IN EFI_EVENT                              TimeoutEvent,
                           IN UINT16                                 OpFlags,
                           IN BOOLEAN *                              IsDone,
                           OUT BOOLEAN *                             IsMatched,
                           IN OUT EFI_IP_ADDRESS *                   DestIp,
                           IN OUT EFI_PXE_BASE_CODE_UDP_PORT *       DestPort,
                           IN OUT EFI_IP_ADDRESS *                   SrcIp,
                           IN OUT EFI_PXE_BASE_CODE_UDP_PORT *       SrcPort
                         )
```

This function is to receive packet using Udp4Read.

Parameters:

[in]	*Udp4*	The pointer to EFI_UDP4_PROTOCOL.
[in]	*Token*	The pointer to EFI_UDP4_COMPLETION_TOKEN.
[in]	*Mode*	The pointer to EFI_PXE_BASE_CODE_MODE.
[in]	*TimeoutEvent*	The event for timeout.
[in]	*OpFlags*	The UDP operation flags.
[in]	*IsDone*	The pointer to the IsDone flag.
[out]	*IsMatched*	The pointer to the IsMatched flag.
[in,out]	*DestIp*	The pointer to the destination address.
[in,out]	*DestPort*	The pointer to the destination port.
[in,out]	*SrcIp*	The pointer to the source address.
[in,out]	*SrcPort*	The pointer to the source port.

Return values:

EFI_SUCCESS	Successfully read the data using Udp4.
Others	Failed to send out data.

References PxeBcCheckByDestIp(), PxeBcCheckByDestPort(), PxeBcCheckByIpFilter(), PxeBcFilterBySrcIp(), and PxeBcFilterBySrcPort().

Referenced by EfiPxeBcUdpRead().

```
EFI_STATUS PxeBcUdp4Write ( IN EFI_UDP4_PROTOCOL *          Udp4,
                            IN EFI_UDP4_SESSION_DATA *      Session,
                            IN EFI_EVENT                    TimeoutEvent,
                            IN EFI_IPv4_ADDRESS *           Gateway,
```

```
                                IN UINTN *                    HeaderSize,
                                IN VOID *                     HeaderPtr,
                                IN UINTN *                    BufferSize,
                                IN VOID *                     BufferPtr
                              )
```

This function is to configure a UDPv4 instance for UdpWrite.

Parameters:

[in]	*Udp4*	The pointer to EFI_UDP4_PROTOCOL.
[in]	*Session*	The pointer to the UDP4 session data.
[in]	*TimeoutEvent*	The event for timeout.
[in]	*Gateway*	The pointer to the gateway address.
[in]	*HeaderSize*	An optional field which may be set to the length of a header at HeaderPtr to be prefixed to the data at BufferPtr.
[in]	*HeaderPtr*	If HeaderSize is not NULL, a pointer to a header to be prefixed to the data at BufferPtr.
[in]	*BufferSize*	A pointer to the size of the data at BufferPtr.
[in]	*BufferPtr*	A pointer to the data to be written.

Return values:

EFI_SUCCESS	Successfully send out data using Udp4Write.
Others	Failed to send out data.

References PxeBcCommonNotify().

Referenced by EfiPxeBcUdpWrite().

```
EFI_STATUS PxeBcUdp6Read ( IN EFI_UDP6_PROTOCOL *                  Udp6,
                           IN EFI_UDP6_COMPLETION_TOKEN *          Token,
                           IN EFI_PXE_BASE_CODE_MODE *             Mode,
                           IN EFI_EVENT                            TimeoutEvent,
                           IN UINT16                               OpFlags,
                           IN BOOLEAN *                            IsDone,
                           OUT BOOLEAN *                           IsMatched,
                           IN OUT EFI_IP_ADDRESS *                 DestIp,
                           IN OUT EFI_PXE_BASE_CODE_UDP_PORT *     DestPort,
                           IN OUT EFI_IP_ADDRESS *                 SrcIp,
                           IN OUT EFI_PXE_BASE_CODE_UDP_PORT *     SrcPort
                         )
```

This function is to receive packets using Udp6Read.

Parameters:

[in]	*Udp6*	The pointer to EFI_UDP6_PROTOCOL.
[in]	*Token*	The pointer to EFI_UDP6_COMPLETION_TOKEN.
[in]	*Mode*	The pointer to EFI_PXE_BASE_CODE_MODE.
[in]	*TimeoutEvent*	The event for timeout.
[in]	*OpFlags*	The UDP operation flags.
[in]	*IsDone*	The pointer to the IsDone flag.

[out]	*IsMatched*	The pointer to the IsMatched flag.
[in,out]	*DestIp*	The pointer to the destination address.
[in,out]	*DestPort*	The pointer to the destination port.
[in,out]	*SrcIp*	The pointer to the source address.
[in,out]	*SrcPort*	The pointer to the source port.

Return values:

EFI_SUCCESS	Successfully read data using Udp6.
Others	Failed to send out data.

References PxeBcCheckByDestIp(), PxeBcCheckByDestPort(), PxeBcCheckByIpFilter(), PxeBcFilterBySrcIp(), and PxeBcFilterBySrcPort().

Referenced by EfiPxeBcUdpRead().

```
EFI_STATUS PxeBcUdp6Write ( IN EFI_UDP6_PROTOCOL *          Udp6,
                            IN EFI_UDP6_SESSION_DATA *      Session,
                            IN EFI_EVENT                    TimeoutEvent,
                            IN UINTN *                      HeaderSize,
                            IN VOID *                       HeaderPtr,
                            IN UINTN *                      BufferSize,
                            IN VOID *                       BufferPtr
                          )
```

This function is to configure a UDPv4 instance for UdpWrite.

Parameters:

[in]	*Udp6*	The pointer to EFI_UDP6_PROTOCOL.
[in]	*Session*	The pointer to the UDP6 session data.
[in]	*TimeoutEvent*	The event for timeout.
[in]	*HeaderSize*	An optional field which may be set to the length of a header at HeaderPtr to be prefixed to the data at BufferPtr.
[in]	*HeaderPtr*	If HeaderSize is not NULL, a pointer to a header to be prefixed to the data at BufferPtr.
[in]	*BufferSize*	A pointer to the size of the data at BufferPtr.
[in]	*BufferPtr*	A pointer to the data to be written.

Return values:

EFI_SUCCESS	Successfully sent out data using Udp6Write.
Others	Failed to send out data.

References PxeBcCommonNotify().

Referenced by EfiPxeBcUdpWrite().

```
UINTN PxeBcUintnToAscDec ( IN UINTN    Number,
                           IN UINT8 *  Buffer,
                           IN UINTN    BufferSize
                         )
```

This function is to convert a UINTN to a ASCII string, and return the actual length of the buffer.

Parameters:

[in]	*Number*	Numeric value to be converted.
[in]	*Buffer*	The pointer to the buffer for ASCII string.
[in]	*BufferSize*	The maxsize of the buffer.

Returns:

Length The actual length of the ASCII string.

Referenced by PxeBcMtftp4GetFileSize(), PxeBcMtftp4ReadDirectory(), PxeBcMtftp4ReadFile(), PxeBcMtftp4WriteFile(), PxeBcMtftp6GetFileSize(), PxeBcMtftp6ReadDirectory(), PxeBcMtftp6ReadFile(), and PxeBcMtftp6WriteFile().

```
VOID PxeBcUintnToAscDecWithFormat ( IN UINTN     Number,
                                    IN UINT8 *   Buffer,
                                    IN INTN      Length
                                  )
```

This function is to convert UINTN to ASCII string with the required formatting.

Parameters:

[in]	*Number*	Numeric value to be converted.
[in]	*Buffer*	The pointer to the buffer for ASCII string.
[in]	*Length*	The length of the required format.

Referenced by PxeBcBuildDhcp4Options(), and PxeBcBuildDhcp6Options().

```
EFI_STATUS PxeBcUniHexToUint8 ( OUT UINT8 *   Digit,
                                IN CHAR16     Char
                              )
```

This function is to convert unicode hex number to a UINT8.

Parameters:

[out]	*Digit*	The converted UINT8 for output.
[in]	*Char*	The unicode hex number to be converted.

Return values:

EFI_SUCCESS	Successfully converted the unicode hex.
EFI_INVALID_PARAMETER	Failed to convert the unicode hex.

Referenced by PxeBcExtractBootFileParam().

NetworkPkg/UefiPxeBcDxe/PxeBcSupport.h File Reference

Defines

"memItemRight" valign="bottom">ICMP_DEST_UNREACHABLE 3

#define ICMP_SOURCE_QUENCH 4
#define ICMP_REDIRECT 5
#define ICMP_ECHO_REQUEST 8
#define ICMP_TIME_EXCEEDED 11
#define ICMP_PARAMETER_PROBLEM 12

Functions

EFI_STATUS PxeBcFlushStationIp (PXEBC_PRIVATE_DATA *Private, EFI_IP_ADDRESS *StationIp, EFI_IP_ADDRESS *SubnetMask)

VOID EFIAPI PxeBcCommonNotify (IN EFI_EVENT Event, IN VOID *Context)

BOOLEAN PxeBcCheckArpCache (IN EFI_PXE_BASE_CODE_MODE *Mode, IN EFI_IPv4_ADDRESS *Ip4Addr, OUT EFI_MAC_ADDRESS *MacAddress)

VOID EFIAPI PxeBcArpCacheUpdate (IN EFI_EVENT Event, IN VOID *Context)

VOID EFIAPI PxeBcIcmpErrorUpdate (IN EFI_EVENT Event, IN VOID *Context)

VOID EFIAPI PxeBcIcmp6ErrorUpdate (IN EFI_EVENT Event, IN VOID *Context)

EFI_STATUS PxeBcConfigUdp4Write (IN EFI_UDP4_PROTOCOL *Udp4, IN EFI_IPv4_ADDRESS *StationIp, IN EFI_IPv4_ADDRESS *SubnetMask, IN EFI_IPv4_ADDRESS *Gateway, IN OUT UINT16 *SrcPort, IN BOOLEAN DoNotFragment)

EFI_STATUS PxeBcConfigUdp6Write (IN EFI_UDP6_PROTOCOL *Udp6, IN EFI_IPv6_ADDRESS *StationIp, IN OUT UINT16 *SrcPort)

EFI_STATUS PxeBcUdp4Write (IN EFI_UDP4_PROTOCOL *Udp4, IN EFI_UDP4_SESSION_DATA *Session, IN EFI_EVENT TimeoutEvent, IN EFI_IPv4_ADDRESS *Gateway, IN

UINTN *HeaderSize, IN VOID *HeaderPtr, IN UINTN *BufferSize, IN VOID *BufferPtr)

EFI_STATUS PxeBcUdp6Write (IN EFI_UDP6_PROTOCOL *Udp6, IN EFI_UDP6_SESSION_DATA *Session, IN EFI_EVENT TimeoutEvent, IN UINTN *HeaderSize, IN VOID *HeaderPtr, IN UINTN *BufferSize, IN VOID *BufferPtr)

BOOLEAN PxeBcCheckByIpFilter (IN EFI_PXE_BASE_CODE_MODE *Mode, IN VOID *Session, IN UINT16 OpFlags)

BOOLEAN PxeBcCheckByDestIp (IN EFI_PXE_BASE_CODE_MODE *Mode, IN VOID *Session, IN OUT EFI_IP_ADDRESS *DestIp, IN UINT16 OpFlags)

BOOLEAN PxeBcCheckByDestPort (IN EFI_PXE_BASE_CODE_MODE *Mode, IN VOID *Session, IN OUT UINT16 *DestPort, IN UINT16 OpFlags)

BOOLEAN PxeBcFilterBySrcIp (IN EFI_PXE_BASE_CODE_MODE *Mode, IN VOID *Session, IN OUT EFI_IP_ADDRESS *SrcIp, IN UINT16 OpFlags)

BOOLEAN PxeBcFilterBySrcPort (IN EFI_PXE_BASE_CODE_MODE *Mode, IN VOID *Session, IN OUT UINT16 *SrcPort, IN UINT16 OpFlags)

EFI_STATUS PxeBcUdp4Read (IN EFI_UDP4_PROTOCOL *Udp4, IN EFI_UDP4_COMPLETION_TOKEN *Token, IN EFI_PXE_BASE_CODE_MODE *Mode, IN EFI_EVENT TimeoutEvent, IN UINT16 OpFlags, IN BOOLEAN *IsDone, OUT BOOLEAN *IsMatched, IN OUT EFI_IP_ADDRESS *DestIp, IN OUT EFI_PXE_BASE_CODE_UDP_PORT *DestPort, IN OUT EFI_IP_ADDRESS *SrcIp, IN OUT EFI_PXE_BASE_CODE_UDP_PORT *SrcPort)

EFI_STATUS PxeBcUdp6Read (IN EFI_UDP6_PROTOCOL *Udp6, IN EFI_UDP6_COMPLETION_TOKEN *Token, IN

	EFI_PXE_BASE_CODE_MODE *Mode, IN EFI_EVENT TimeoutEvent, IN UINT16 OpFlags, IN BOOLEAN *IsDone, OUT BOOLEAN *IsMatched, IN OUT EFI_IP_ADDRESS *DestIp, IN OUT EFI_PXE_BASE_CODE_UDP_PORT *DestPort, IN OUT EFI_IP_ADDRESS *SrcIp, IN OUT EFI_PXE_BASE_CODE_UDP_PORT *SrcPort)
VOID	PxeBcShowIp4Addr (IN EFI_IPv4_ADDRESS *Ip)
VOID	PxeBcShowIp6Addr (IN EFI_IPv6_ADDRESS *Ip)
VOID	PxeBcUintnToAscDecWithFormat (IN UINTN Number, IN UINT8 *Buffer, IN INTN Length)
UINTN	PxeBcUintnToAscDec (IN UINTN Number, IN UINT8 *Buffer, IN UINTN BufferSize)
EFI_STATUS	PxeBcUniHexToUint8 (OUT UINT8 *Digit, IN CHAR16 Char)
VOID	CalcElapsedTime (IN PXEBC_PRIVATE_DATA *Private)
EFI_HANDLE	PxeBcGetNicByIp4Children (IN EFI_HANDLE ControllerHandle)
EFI_HANDLE	PxeBcGetNicByIp6Children (IN EFI_HANDLE ControllerHandle)

Detailed Description

Support functions declaration for UefiPxeBc Driver.

Copyright (c) 2007 - 2014, Intel Corporation. All rights reserved.

This program and the accompanying materials are licensed and made available under the terms and conditions of the BSD License which accompanies this distribution. The full text of the license may be found at http://opensource.org/licenses/bsd-license.php.

THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

Define Documentation

"memdoc">

Referenced by PxeBcIcmpErrorDpcHandle().

#define ICMP_ECHO_REQUEST 8

#define ICMP_PARAMETER_PROBLEM 12

Referenced by PxeBcIcmpErrorDpcHandle().

#define ICMP_REDIRECT 5

Referenced by PxeBcIcmpErrorDpcHandle().

#define ICMP_SOURCE_QUENCH 4

Referenced by PxeBcIcmpErrorDpcHandle().

#define ICMP_TIME_EXCEEDED 11

Referenced by PxeBcIcmpErrorDpcHandle().

Function Documentation

VOID CalcElapsedTime (IN PXEBC_PRIVATE_DATA * *Private*)

Calculate the elapsed time.

Parameters:

[in] *Private* The pointer to PXE private data

Referenced by PxeBcDhcp6CallBack(), and PxeBcRequestBootService().

VOID EFIAPI PxeBcArpCacheUpdate (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

Update arp cache periodically.

Parameters:

Event Pointer to EFI_PXE_BC_PROTOCOL.
Context Context of the timer event.

Update the arp cache periodically.

Parameters:

Event The pointer to EFI_PXE_BC_PROTOCOL.
Context Context of the timer event.

References _PXEBC_PRIVATE_DATA::Arp, and _PXEBC_PRIVATE_DATA::PxeBc.

Referenced by EfiPxeBcArp(), and EfiPxeBcStart().

BOOLEAN PxeBcCheckArpCache (IN EFI_PXE_BASE_CODE_MODE * *Mode*,

```
                                    IN EFI_IPv4_ADDRESS *          Ip4Addr,
                                    OUT EFI_MAC_ADDRESS *          MacAddress
                                  )
```

Perform arp resolution from the arp cache in PxeBcMode.

Parameters:

Mode	Pointer to EFI_PXE_BASE_CODE_MODE.
Ip4Addr	The Ip4 address for resolution.
MacAddress	The resoluted MAC address if the resolution is successful. The value is undefined if resolution fails.

Return values:

TRUE	Found a matched entry.
FALSE	Did not find a matched entry.

Do arp resolution from arp cache in PxeBcMode.

Parameters:

Mode	The pointer to EFI_PXE_BASE_CODE_MODE.
Ip4Addr	The Ip4 address for resolution.
MacAddress	The resoluted MAC address if the resolution is successful. The value is undefined if the resolution fails.

Return values:

TRUE	Found an matched entry.
FALSE	Did not find a matched entry.

Referenced by EfiPxeBcArp().

```
BOOLEAN PxeBcCheckByDestIp ( IN EFI_PXE_BASE_CODE_MODE *  Mode,
                             IN VOID *                    Session,
                             IN OUT EFI_IP_ADDRESS *      DestIp,
                             IN UINT16                    OpFlags
                           )
```

Filter the received packet with the destination Ip.

Parameters:

`[in]`	*Mode*	Pointer to mode data of PxeBc.
`[in]`	*Session*	Pointer to the current UDPv4 session.
`[in,out]`	*DestIp*	Pointer to the dest Ip address.
`[in]`	*OpFlags*	Operation flag for UdpRead/UdpWrite.

Return values:

TRUE	Succesfully passed the IPv4 filter.
FALSE	Failed to pass the IPv4 filter.

Filter the received packet using the destination Ip.

Parameters:

`[in]`	*Mode*	The pointer to the mode data of PxeBc.
`[in]`	*Session*	The pointer to the current UDPv4 session.

[in,out]	*DestIp*	The pointer to the destination Ip address.
[in]	*OpFlags*	Operation flag for UdpRead/UdpWrite.

Return values:

TRUE	Passed the IPv4 filter successfully.
FALSE	Failed to pass the IPv4 filter.

Referenced by PxeBcUdp4Read(), and PxeBcUdp6Read().

```
BOOLEAN PxeBcCheckByDestPort ( IN EFI_PXE_BASE_CODE_MODE *  Mode,
                               IN VOID *                    Session,
                               IN OUT UINT16 *              DestPort,
                               IN UINT16                    OpFlags
                             )
```

Check the received packet with the destination port.

Parameters:

[in]	*Mode*	Pointer to mode data of PxeBc.
[in]	*Session*	Pointer to the current UDPv4 session.
[in,out]	*DestPort*	Pointer to the destination port.
[in]	*OpFlags*	Operation flag for UdpRead/UdpWrite.

Return values:

TRUE	Succesfully passed the IPv4 filter.
FALSE	Failed to pass the IPv4 filter.

Check the received packet using the destination port.

Parameters:

[in]	*Mode*	The pointer to the mode data of PxeBc.
[in]	*Session*	The pointer to the current UDPv4 session.
[in,out]	*DestPort*	The pointer to the destination port.
[in]	*OpFlags*	Operation flag for UdpRead/UdpWrite.

Return values:

TRUE	Passed the IPv4 filter successfully.
FALSE	Failed to pass the IPv4 filter.

Referenced by PxeBcUdp4Read(), and PxeBcUdp6Read().

```
BOOLEAN PxeBcCheckByIpFilter ( IN EFI_PXE_BASE_CODE_MODE *  Mode,
                               IN VOID *                    Session,
                               IN UINT16                    OpFlags
                             )
```

Check the received packet with the Ip filter.

Parameters:

[in]	*Mode*	Pointer to mode data of PxeBc.
[in]	*Session*	Pointer to the current UDPv4 session.

[in] *OpFlags* Operation flag for UdpRead/UdpWrite.

Return values:

TRUE Succesfully passed the Ip filter.
FALSE Failed to pass the Ip filter.

Check the received packet using the Ip filter.

Parameters:

[in] *Mode* The pointer to the mode data of PxeBc.
[in] *Session* The pointer to the current UDPv4 session.
[in] *OpFlags* Operation flag for UdpRead/UdpWrite.

Return values:

TRUE Passed the Ip filter successfully.
FALSE Failed to pass the Ip filter.

Referenced by PxeBcUdp4Read(), and PxeBcUdp6Read().

```
VOID EFIAPI PxeBcCommonNotify ( IN EFI_EVENT   Event,
                                IN VOID *      Context
                              )
```

Notify callback function when an event is triggered.

Parameters:

[in] *Event* The triggered event.
[in] *Context* The opaque parameter to the function.

Notify the callback function when an event is triggered.

Parameters:

[in] *Event* The triggered event.
[in] *Context* The opaque parameter to the function.

Referenced by EfiPxeBcArp(), EfiPxeBcUdpRead(), PxeBcRegisterIp6Address(), PxeBcUdp4Write(), and PxeBcUdp6Write().

```
EFI_STATUS PxeBcConfigUdp4Write ( IN EFI_UDP4_PROTOCOL *   Udp4,
                                  IN EFI_IPv4_ADDRESS *    StationIp,
                                  IN EFI_IPv4_ADDRESS *    SubnetMask,
                                  IN EFI_IPv4_ADDRESS *    Gateway,
                                  IN OUT UINT16 *          SrcPort,
                                  IN BOOLEAN               DoNotFragment
                                )
```

This function is to configure a UDPv4 instance for UdpWrite.

Parameters:

[in] *Udp4* Pointer to EFI_UDP4_PROTOCOL.
[in] *StationIp* Pointer to the station address.

[in]	*SubnetMask*	Pointer to the subnet mask.
[in]	*Gateway*	Pointer to the gateway address.
[in,out]	*SrcPort*	Pointer to the source port.
[in]	*DoNotFragment*	The flag of DoNotFragment bit in the IPv4 packet.

Return values:

EFI_SUCCESS	Successfully configured this instance.
Others	Failed to configure this instance.

This function is to configure a UDPv4 instance for UdpWrite.

Parameters:

[in]	*Udp4*	The pointer to EFI_UDP4_PROTOCOL.
[in]	*StationIp*	The pointer to the station address.
[in]	*SubnetMask*	The pointer to the subnet mask.
[in]	*Gateway*	The pointer to the gateway address.
[in,out]	*SrcPort*	The pointer to the source port.
[in]	*DoNotFragment*	If TRUE, fragment is not enabled. Otherwise, fragment is enabled.

Return values:

EFI_SUCCESS	Successfully configured this instance.
Others	Failed to configure this instance.

References PXEBC_DEFAULT_LIFETIME.

Referenced by EfiPxeBcUdpWrite().

```
EFI_STATUS PxeBcConfigUdp6Write ( IN EFI_UDP6_PROTOCOL *    Udp6,
                                  IN EFI_IPv6_ADDRESS *     StationIp,
                                  IN OUT UINT16 *           SrcPort
                                )
```

This function is to configure a UDPv6 instance for UdpWrite.

Parameters:

[in]	*Udp6*	Pointer to EFI_UDP6_PROTOCOL.
[in]	*StationIp*	Pointer to the station address.
[in,out]	*SrcPort*	Pointer to the source port.

Return values:

EFI_SUCCESS	Successfuly configured this instance.
Others	Failed to configure this instance.

This function is to configure a UDPv6 instance for UdpWrite.

Parameters:

[in]	*Udp6*	The pointer to EFI_UDP6_PROTOCOL.
[in]	*StationIp*	The pointer to the station address.
[in,out]	*SrcPort*	The pointer to the source port.

Return values:

EFI_SUCCESS	Successfully configured this instance.

Others Failed to configure this instance.

References PXEBC_DEFAULT_HOPLIMIT, and PXEBC_DEFAULT_LIFETIME.

Referenced by EfiPxeBcUdpWrite().

```
BOOLEAN PxeBcFilterBySrcIp ( IN EFI_PXE_BASE_CODE_MODE *  Mode,
                             IN VOID *                    Session,
                             IN OUT EFI_IP_ADDRESS *      SrcIp,
                             IN UINT16                    OpFlags
                           )
```

Filter the received packet with the source Ip.

Parameters:

[in]	*Mode*	Pointer to mode data of PxeBc.
[in]	*Session*	Pointer to the current UDPv4 session.
[in,out]	*SrcIp*	Pointer to the source Ip address.
[in]	*OpFlags*	Operation flag for UdpRead/UdpWrite.

Return values:

TRUE Succesfully passed the IPv4 filter.
FALSE Failed to pass the IPv4 filter.

Filter the received packet using the source Ip.

Parameters:

[in]	*Mode*	The pointer to the mode data of PxeBc.
[in]	*Session*	The pointer to the current UDPv4 session.
[in,out]	*SrcIp*	The pointer to the source Ip address.
[in]	*OpFlags*	Operation flag for UdpRead/UdpWrite.

Return values:

TRUE Passed the IPv4 filter successfully.
FALSE Failed to pass the IPv4 filter.

Referenced by PxeBcUdp4Read(), and PxeBcUdp6Read().

```
BOOLEAN PxeBcFilterBySrcPort ( IN EFI_PXE_BASE_CODE_MODE *  Mode,
                               IN VOID *                    Session,
                               IN OUT UINT16 *              SrcPort,
                               IN UINT16                    OpFlags
                             )
```

Filter the received packet with the source port.

Parameters:

[in]	*Mode*	Pointer to mode data of PxeBc.
[in]	*Session*	Pointer to the current UDPv4 session.
[in,out]	*SrcPort*	Pointer to the source port.
[in]	*OpFlags*	Operation flag for UdpRead/UdpWrite.

Return values:

TRUE Succesfully passed the IPv4 filter.
FALSE Failed to pass the IPv4 filter.

Filter the received packet using the source port.

Parameters:

[in]	*Mode*	The pointer to the mode data of PxeBc.
[in]	*Session*	The pointer to the current UDPv4 session.
[in,out]	*SrcPort*	The pointer to the source port.
[in]	*OpFlags*	Operation flag for UdpRead/UdpWrite.

Return values:

TRUE Passed the IPv4 filter successfully.
FALSE Failed to pass the IPv4 filter.

Referenced by PxeBcUdp4Read(), and PxeBcUdp6Read().

```
EFI_STATUS PxeBcFlushStationIp ( PXEBC_PRIVATE_DATA *  Private,
                                 EFI_IP_ADDRESS *      StationIp,
                                 EFI_IP_ADDRESS *      SubnetMask
                               )
```

Flush the previous configration using the new station Ip address.

Parameters:

[in]	*Private*	Pointer to PxeBc private data.
[in]	*StationIp*	Pointer to the station Ip address.
[in]	*SubnetMask*	Pointer to the subnet mask address for v4.

Return values:

EFI_SUCCESS Successfully flushed the previous config.
Others Failed to flush using the new station Ip.

Flush the previous configration using the new station Ip address.

Parameters:

[in]	*Private*	The pointer to the PxeBc private data.
[in]	*StationIp*	The pointer to the station Ip address.
[in]	*SubnetMask*	The pointer to the subnet mask address for v4.

Return values:

EFI_SUCCESS Successfully flushed the previous configuration.
Others Failed to flush using the new station Ip.

References _PXEBC_PRIVATE_DATA::Icmp6Token, _PXEBC_PRIVATE_DATA::IcmpToken, _PXEBC_PRIVATE_DATA::Ip4, _PXEBC_PRIVATE_DATA::Ip4CfgData, _PXEBC_PRIVATE_DATA::Ip6, _PXEBC_PRIVATE_DATA::Ip6CfgData, _PXEBC_PRIVATE_DATA::PxeBc, _PXEBC_PRIVATE_DATA::Udp4CfgData, and _PXEBC_PRIVATE_DATA::Udp6CfgData.

Referenced by EfiPxeBcSetStationIP(), PxeBcDhcp4Dora(), and PxeBcSetIp6Address().

EFI_HANDLE PxeBcGetNicByIp4Children (IN EFI_HANDLE *ControllerHandle*)
Get the Nic handle using any child handle in the IPv4 stack.

Parameters:
`[in]` *ControllerHandle* Pointer to child handle over IPv4.

Returns:
NicHandle The pointer to the Nic handle.

Referenced by PxeBcComponentNameGetControllerName(), and PxeBcStop().

EFI_HANDLE PxeBcGetNicByIp6Children (IN EFI_HANDLE *ControllerHandle*)
Get the Nic handle using any child handle in the IPv6 stack.

Parameters:
`[in]` *ControllerHandle* Pointer to child handle over IPv6.

Returns:
NicHandle The pointer to the Nic handle.

Referenced by PxeBcComponentNameGetControllerName(), and PxeBcStop().

VOID EFIAPI PxeBcIcmp6ErrorUpdate (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

xxx

Parameters:
Event The event signaled.
Context The context passed in by the event notifier.

Callback function to update the latest ICMP6 error message.

Parameters:
Event The event signalled.
Context The context passed in using the event notifier.

References PxeBcIcmp6ErrorDpcHandle().

Referenced by EfiPxeBcStart().

VOID EFIAPI PxeBcIcmpErrorUpdate (IN EFI_EVENT *Event*,
IN VOID * *Context*
)

xxx

Parameters:
Event The event signaled.
Context The context passed in by the event notifier.

Callback function to update the latest ICMP6 error message.

Parameters:

Event The event signalled.

Context The context passed in using the event notifier.

References PxeBcIcmpErrorDpcHandle().

Referenced by EfiPxeBcStart().

VOID PxeBcShowIp4Addr (IN EFI_IPv4_ADDRESS * *Ip*)

This function is to display the IPv4 address.

Parameters:

[in] *Ip* Pointer to the IPv4 address.

This function is to display the IPv4 address.

Parameters:

[in] *Ip* The pointer to the IPv4 address.

Referenced by PxeBcDhcp4BootInfo(), and PxeBcDhcp4Dora().

VOID PxeBcShowIp6Addr (IN EFI_IPv6_ADDRESS * *Ip*)

This function is to display the IPv6 address.

Parameters:

[in] *Ip* Pointer to the IPv6 address.

This function is to display the IPv6 address.

Parameters:

[in] *Ip* The pointer to the IPv6 address.

Referenced by PxeBcDhcp6BootInfo(), and PxeBcSetIp6Address().

```
EFI_STATUS PxeBcUdp4Read ( IN EFI_UDP4_PROTOCOL *                  Udp4,
                           IN EFI_UDP4_COMPLETION_TOKEN *          Token,
                           IN EFI_PXE_BASE_CODE_MODE *             Mode,
                           IN EFI_EVENT                            TimeoutEvent,
                           IN UINT16                               OpFlags,
                           IN BOOLEAN *                            IsDone,
                           OUT BOOLEAN *                           IsMatched,
                           IN OUT EFI_IP_ADDRESS *                 DestIp,
                           IN OUT EFI_PXE_BASE_CODE_UDP_PORT *     DestPort,
                           IN OUT EFI_IP_ADDRESS *                 SrcIp,
                           IN OUT EFI_PXE_BASE_CODE_UDP_PORT *     SrcPort
                         )
```

This function is to receive packet with Udp4Read.

Parameters:

[in]	*Udp4*	Pointer to EFI_UDP4_PROTOCOL.
[in]	*Token*	Pointer to EFI_UDP4_COMPLETION_TOKEN.
[in]	*Mode*	Pointer to EFI_PXE_BASE_CODE_MODE.
[in]	*TimeoutEvent*	The event for timeout.
[in]	*OpFlags*	The UDP operation flags.
[in]	*IsDone*	Pointer to IsDone flag.
[out]	*IsMatched*	Pointer to IsMatched flag.
[in,out]	*DestIp*	Pointer to destination address.
[in,out]	*DestPort*	Pointer to destination port.
[in,out]	*SrcIp*	Pointer to source address.
[in,out]	*SrcPort*	Pointer to source port.

Return values:

EFI_SUCCESS	Successfully read data with Udp4.
Others	Failed to send out data.

This function is to receive packet using Udp4Read.

Parameters:

[in]	*Udp4*	The pointer to EFI_UDP4_PROTOCOL.
[in]	*Token*	The pointer to EFI_UDP4_COMPLETION_TOKEN.
[in]	*Mode*	The pointer to EFI_PXE_BASE_CODE_MODE.
[in]	*TimeoutEvent*	The event for timeout.
[in]	*OpFlags*	The UDP operation flags.
[in]	*IsDone*	The pointer to the IsDone flag.
[out]	*IsMatched*	The pointer to the IsMatched flag.
[in,out]	*DestIp*	The pointer to the destination address.
[in,out]	*DestPort*	The pointer to the destination port.
[in,out]	*SrcIp*	The pointer to the source address.
[in,out]	*SrcPort*	The pointer to the source port.

Return values:

EFI_SUCCESS	Successfully read the data using Udp4.
Others	Failed to send out data.

References PxeBcCheckByDestIp(), PxeBcCheckByDestPort(), PxeBcCheckByIpFilter(), PxeBcFilterBySrcIp(), and PxeBcFilterBySrcPort().

Referenced by EfiPxeBcUdpRead().

```
EFI_STATUS PxeBcUdp4Write ( IN EFI_UDP4_PROTOCOL *          Udp4,
                            IN EFI_UDP4_SESSION_DATA *      Session,
                            IN EFI_EVENT                    TimeoutEvent,
                            IN EFI_IPv4_ADDRESS *           Gateway,
                            IN UINTN *                      HeaderSize,
                            IN VOID *                       HeaderPtr,
                            IN UINTN *                      BufferSize,
```

```
                                IN VOID *                    BufferPtr
                              )
```

This function is to configure a UDPv4 instance for UdpWrite.

Parameters:

[in]	*Udp4*	Pointer to EFI_UDP4_PROTOCOL.
[in]	*Session*	Pointer to the UDP4 session data.
[in]	*TimeoutEvent*	The event for timeout.
[in]	*Gateway*	Pointer to the gateway address.
[in]	*HeaderSize*	An optional field which may be set to the length of a header at HeaderPtr to be prefixed to the data at BufferPtr.
[in]	*HeaderPtr*	If HeaderSize is not NULL, a pointer to a header to be prefixed to the data at BufferPtr.
[in]	*BufferSize*	A pointer to the size of the data at BufferPtr.
[in]	*BufferPtr*	A pointer to the data to be written.

Return values:

EFI_SUCCESS	Successfully sent out data with Udp4Write.
Others	Failed to send out data.

This function is to configure a UDPv4 instance for UdpWrite.

Parameters:

[in]	*Udp4*	The pointer to EFI_UDP4_PROTOCOL.
[in]	*Session*	The pointer to the UDP4 session data.
[in]	*TimeoutEvent*	The event for timeout.
[in]	*Gateway*	The pointer to the gateway address.
[in]	*HeaderSize*	An optional field which may be set to the length of a header at HeaderPtr to be prefixed to the data at BufferPtr.
[in]	*HeaderPtr*	If HeaderSize is not NULL, a pointer to a header to be prefixed to the data at BufferPtr.
[in]	*BufferSize*	A pointer to the size of the data at BufferPtr.
[in]	*BufferPtr*	A pointer to the data to be written.

Return values:

EFI_SUCCESS	Successfully send out data using Udp4Write.
Others	Failed to send out data.

References PxeBcCommonNotify().

Referenced by EfiPxeBcUdpWrite().

```
EFI_STATUS PxeBcUdp6Read ( IN EFI_UDP6_PROTOCOL *                Udp6,
                           IN EFI_UDP6_COMPLETION_TOKEN *        Token,
                           IN EFI_PXE_BASE_CODE_MODE *           Mode,
                           IN EFI_EVENT                          TimeoutEvent,
                           IN UINT16                             OpFlags,
                           IN BOOLEAN *                          IsDone,
                           OUT BOOLEAN *                         IsMatched,
                           IN OUT EFI_IP_ADDRESS *               DestIp,
```

```
                                IN OUT EFI_PXE_BASE_CODE_UDP_PORT *   DestPort,
                                IN OUT EFI_IP_ADDRESS *               SrcIp,
                                IN OUT EFI_PXE_BASE_CODE_UDP_PORT *   SrcPort
                              )
```

This function is to receive packet with Udp6Read.

Parameters:

[in]	*Udp6*	Pointer to EFI_UDP6_PROTOCOL.
[in]	*Token*	Pointer to EFI_UDP6_COMPLETION_TOKEN.
[in]	*Mode*	Pointer to EFI_PXE_BASE_CODE_MODE.
[in]	*TimeoutEvent*	The event for timeout.
[in]	*OpFlags*	The UDP operation flags.
[in]	*IsDone*	Pointer to IsDone flag.
[out]	*IsMatched*	Pointer to IsMatched flag.
[in,out]	*DestIp*	Pointer to destination address.
[in,out]	*DestPort*	Pointer to destination port.
[in,out]	*SrcIp*	Pointer to source address.
[in,out]	*SrcPort*	Pointer to source port.

Return values:

EFI_SUCCESS	Successfully read data with Udp6.
Others	Failed to send out data.

This function is to receive packets using Udp6Read.

Parameters:

[in]	*Udp6*	The pointer to EFI_UDP6_PROTOCOL.
[in]	*Token*	The pointer to EFI_UDP6_COMPLETION_TOKEN.
[in]	*Mode*	The pointer to EFI_PXE_BASE_CODE_MODE.
[in]	*TimeoutEvent*	The event for timeout.
[in]	*OpFlags*	The UDP operation flags.
[in]	*IsDone*	The pointer to the IsDone flag.
[out]	*IsMatched*	The pointer to the IsMatched flag.
[in,out]	*DestIp*	The pointer to the destination address.
[in,out]	*DestPort*	The pointer to the destination port.
[in,out]	*SrcIp*	The pointer to the source address.
[in,out]	*SrcPort*	The pointer to the source port.

Return values:

EFI_SUCCESS	Successfully read data using Udp6.
Others	Failed to send out data.

References PxeBcCheckByDestIp(), PxeBcCheckByDestPort(), PxeBcCheckByIpFilter(), PxeBcFilterBySrcIp(), and PxeBcFilterBySrcPort().

Referenced by EfiPxeBcUdpRead().

```
EFI_STATUS PxeBcUdp6Write ( IN EFI_UDP6_PROTOCOL *          Udp6,
                            IN EFI_UDP6_SESSION_DATA *      Session,
                            IN EFI_EVENT                    TimeoutEvent,
```

```
                    IN UINTN *          HeaderSize,
                    IN VOID *           HeaderPtr,
                    IN UINTN *          BufferSize,
                    IN VOID *           BufferPtr
                  )
```

This function is to configure a UDPv6 instance for UdpWrite.

Parameters:

[in]	*Udp6*	Pointer to EFI_UDP6_PROTOCOL.
[in]	*Session*	Pointer to the UDP6 session data.
[in]	*TimeoutEvent*	The event for timeout.
[in]	*HeaderSize*	An optional field which may be set to the length of a header at HeaderPtr to be prefixed to the data at BufferPtr.
[in]	*HeaderPtr*	If HeaderSize is not NULL, a pointer to a header to be prefixed to the data at BufferPtr.
[in]	*BufferSize*	A pointer to the size of the data at BufferPtr.
[in]	*BufferPtr*	A pointer to the data to be written.

Return values:

EFI_SUCCESS	Successfully to send out data with Udp6Write.
Others	Failed to send out data.

This function is to configure a UDPv4 instance for UdpWrite.

Parameters:

[in]	*Udp6*	The pointer to EFI_UDP6_PROTOCOL.
[in]	*Session*	The pointer to the UDP6 session data.
[in]	*TimeoutEvent*	The event for timeout.
[in]	*HeaderSize*	An optional field which may be set to the length of a header at HeaderPtr to be prefixed to the data at BufferPtr.
[in]	*HeaderPtr*	If HeaderSize is not NULL, a pointer to a header to be prefixed to the data at BufferPtr.
[in]	*BufferSize*	A pointer to the size of the data at BufferPtr.
[in]	*BufferPtr*	A pointer to the data to be written.

Return values:

EFI_SUCCESS	Successfully sent out data using Udp6Write.
Others	Failed to send out data.

References PxeBcCommonNotify().

Referenced by EfiPxeBcUdpWrite().

```
UINTN PxeBcUintnToAscDec ( IN UINTN     Number,
                           IN UINT8 *   Buffer,
                           IN UINTN     BufferSize
                         )
```

This function is to convert a UINTN to a ASCII string, and return the actual length of the buffer.

Parameters:

[in] *Number* Numeric value to be converted.
[in] *Buffer* Pointer to the buffer for ASCII string.
[in] *BufferSize* The maxsize of the buffer.

Returns:
Length The actual length of the ASCII string.

This function is to convert a UINTN to a ASCII string, and return the actual length of the buffer.

Parameters:
[in] *Number* Numeric value to be converted.
[in] *Buffer* The pointer to the buffer for ASCII string.
[in] *BufferSize* The maxsize of the buffer.

Returns:
Length The actual length of the ASCII string.

Referenced by PxeBcMtftp4GetFileSize(), PxeBcMtftp4ReadDirectory(), PxeBcMtftp4ReadFile(), PxeBcMtftp4WriteFile(), PxeBcMtftp6GetFileSize(), PxeBcMtftp6ReadDirectory(), PxeBcMtftp6ReadFile(), and PxeBcMtftp6WriteFile().

```
VOID PxeBcUintnToAscDecWithFormat ( IN UINTN    Number,
                                    IN UINT8 *  Buffer,
                                    IN INTN     Length
                                  )
```

This function is to convert UINTN to ASCII string with required format.

Parameters:
[in] *Number* Numeric value to be converted.
[in] *Buffer* Pointer to the buffer for ASCII string.
[in] *Length* Length of the required format.

This function is to convert UINTN to ASCII string with the required formatting.

Parameters:
[in] *Number* Numeric value to be converted.
[in] *Buffer* The pointer to the buffer for ASCII string.
[in] *Length* The length of the required format.

Referenced by PxeBcBuildDhcp4Options(), and PxeBcBuildDhcp6Options().

```
EFI_STATUS PxeBcUniHexToUint8 ( OUT UINT8 *  Digit,
                                IN CHAR16    Char
                              )
```

This function is to convert unicode hex number to a UINT8.

Parameters:
[out] *Digit* The converted UINT8 for output.
[in] *Char* The unicode hex number to be converted.

Return values:

EFI_SUCCESS	Successfully converted the unicode hex.
EFI_INVALID_PARAMETER	Failed to convert the unicode hex.

Referenced by PxeBcExtractBootFileParam().

Generated on Mon Sep 28 08:49:57 2015 for NetworkPkg[ALL] by 1.5.7.1

- Main Page
- Data Structures
- Files

- File List
- Globals

- All
- Functions
- Variables
- Typedefs
- Enumerations
- Enumerator
- Defines

- a
- b
- c
- d
- e
- f
- g
- h
- i
- k
- l
- m
- n
- o
- p
- q
- r
- s
- t
- u
- v

Here is a list of all functions, variables, defines, enums, and typedefs with links to the files they belong to:

- a -

- ACTION : PolicyEntryOperation.h
- AddDns4ServerIp() : DnsImpl.h , DnsImpl.c
- AddDns6ServerIp() : DnsImpl.h , DnsImpl.c
- AddOrInsertPolicyEntry() : PolicyEntryOperation.c , PolicyEntryOperation.h
- ADDRESS_STR_MAX_SIZE : Ip6NvData.h

- ADDRESS_STR_MIN_SIZE : Ip6NvData.h
- AddVlan() :