Customizing a NIC Driver Using the ODX Protocol

May 2000. Software Release SDK 3.0
Document Revision 1.0
Part Number A22384

Intel Corporation
1350 Villa Street
Mountain View, CA 94041-1126
Tel: 650.567.9800
Fax: 650.567.9810
www.intel.com

Contents

About This Guide

This guide describes the Intel® Optimal Data Exchange (ODX) Protocol and explains how to use it. The ODX Protocol allows you to customize a driver for a network interface card (NIC) so that an Intel Policy Accelerator can use the NIC for packet traffic in the same way that it uses its own internal network interfaces.

This is useful in cases where a NIC provides a different kind of cable interface from those provided on the Policy Accelerator.

This allows packet-processing applications written using the Intel IX-API Software Development Kit (SDK) to use the NIC's interface in the same way that it uses the internal Policy Accelerator interfaces.

Audience

This guide is intended for NIC driver customizers who need to modify a NIC driver to interact with a Policy Accelerator. It assumes that you are familiar with the following:

- C programming
- Driver code for the NIC driver to be customized
- Basic concepts of Windows NT device driver development
- The content of the Windows NT DDK help
- The Intel Internet Exchange™ API (IX-API) Software Development Kit (SDK) sufficiently to test the NIC driver

In This Guide

This guide includes the following chapters:

- **Chapter 1, "Introduction to the ODX Protocol,"** which provides background on the Intel IX-API SDK, IX-API, and Policy Accelerator; introduces the ODX Protocol; and explains how they are related
- **Chapter 2, "Customizing a NIC Driver,"** which explains the concepts behind the ODX Protocol and lists the steps that you must take to implement the NIC driver's half of the ODX functions
- **Chapter 3, "Alphabetic Reference: NBFIF (NIC Driver) Functions, Types, and Structures,"** which contains detailed descriptions of each of the functions that you must implement and of the data elements that the functions use
- **Chapter 4, "Alphabetic Reference: NBPE (PA-100 Driver) Functions, Types, and Structures,"** which contains detailed descriptions of each of the functions that your NIC driver calls as part of its customization and of the data elements that the functions use

Other Sources of Information

This guide is part of the Intel® IX-API SDK documentation set, which also includes:

- *IX-API SDK Reference,* which describes the Intel IX-API SDK (software development kit)
- *Developing Applications Using the IX-API SDK,* which provides programmers with conceptual descriptions and instructions on writing network policy-enforcement applications using the SDK
- *IX-API SDK Release Notes,* which lists information about the latest software release
- *Installing the IX-API SDK,* which describes how to install both the run-time and the development versions of the SDK
- *Installing a Policy Accelerator 100 Board,* which describes how to install a Policy Accelerator PCI board into a PC

In addition, the Intel Web site provides valuable information on products, support, and the company. See "Contacting Intel" on page x.

Typographic Conventions

This document uses the following typographic conventions to help you locate and identify information:

Italic text	Used for new terms, emphasis, and book titles; also identifies arguments in syntax descriptions.
Bold text	Identifies keywords and punctuation in syntax descriptions.
`Courier font`	Identifies file names, folder names, and text that either appears on the screen or that you are required to type.

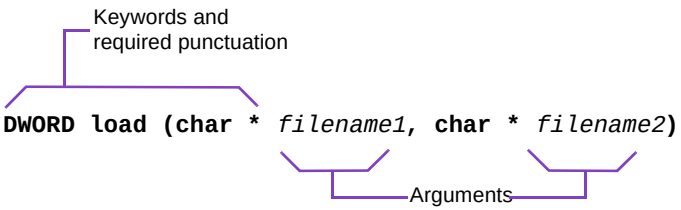
NOTE: Provides extra information, tips, and hints regarding the topic.

CAUTION: Identifies important information about actions that could result in damage to or loss of data or could cause the application to behave in unexpected ways.

WARNING! Identifies critical information about actions that could result in equipment failure or bodily injury.

Syntax Examples

The following figure shows a sample syntax notation.



Installation Path

The following notation is a place holder for the complete pathname of the IX-API SDK installation directory:

SDKinstallpath/

Contacting Intel

You can reach Intel's automated support services 24 hours a day, every day at no charge. The services contain the most up-to-date information about Intel products. You can access installation instructions, troubleshooting information, and general product information.

Web and Internet Sites

You can use the internet to download software updates, troubleshooting tips, installation notes, and more.

- General online support services are on the World Wide Web at:

  `http://support.intel.com`

- Online support services for the Policy Accelerator 100 are on the World Wide Web at:

  `http://support.intel.com/support/network/adapter/pa/pa100/`

For specific types of information and services, go to the following Web and internet sites:

- **Corporate:** `http://www.intel.com`
- **Network Products:** `http://www.intel.com/network`
- **Intel IX Information:** `http://developer.intel.com/design/ixa/`
- **IX-API SDK:** `http://developer.intel.com/design/ixa/software/index.htm`
- **Policy Accelerator:** `http://developer.intel.com/design/ixa/pa100/index.htm`
- **ASIC:** `http://128.11.21.45/scripts/mardev/product/ixe100.asp`
- **FTP Host:** `download.intel.com`
- **FTP Directory:** `/support/network/adapter`

Customer Support Technicians

- **United States and Canada:** 1-916-377-7000 (7:00 - 17:00 M-F Pacific Time)
- **Worldwide Access:** Intel has technical support centers worldwide. Many of the centers are staffed by technicians who speak the local languages. For a list of all Intel support centers, their telephone numbers, and the times they are open, go to:

  `http://support.intel.com/support/9089.htm`

Chapter 1
Introduction to the ODX Protocol

This chapter provides an overview of the Intel® Optimal Data Exchange (ODX) Protocol, its elements, and how to use it.

It contains the following sections:

- Background
- About the ODX Protocol
- Connecting the NIC and the Policy Accelerator
- What the ODX Protocol Includes
- How Applications Can Use the New Driver

Background

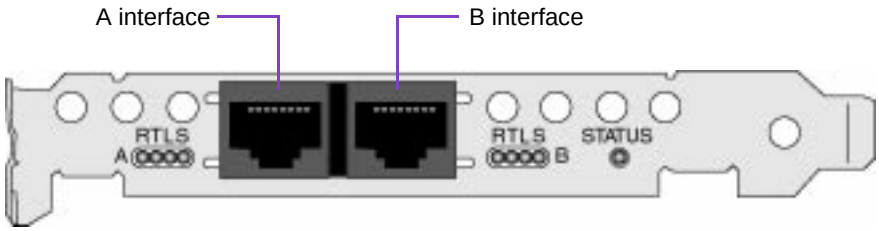
The ODX Protocol allows you to customize a driver for a network interface card (NIC) so that an Intel Policy Accelerator can use the NIC for packet traffic in the same way that it uses its internal network interfaces.

Policy Accelerator Boards

The Intel Policy Accelerator is a wire-speed board designed specifically for enforcing network policies regarding packet flow. It does this by classifying packets, performing actions upon them, and disposing of them according to applications that you, or other third parties, create.

A host system can contain one or more Policy Accelerators; each Policy Accelerator can support multiple applications.

Each Policy Accelerator contains two network interfaces, named A and B, similar to those shown in the following diagram:



The Policy Accelerator communicates with its host across the PCI bus.

For specifications for the Policy Accelerator and its network interfaces, see the document *Installing a Policy Accelerator 100 Board.*

IX-API SDK and IX-API

Developers of network-policy applications use the Intel Internet Exchange™ Software Development Kit (the *IX-API SDK*) application programming interface (the IX-API) to specify the flow of packets through the Policy Accelerator and the operations performed on them.

Such applications are known as *IXA applications*.

The IX-API allows applications to direct packets through the Policy Accelerator by programmatically *binding* the network interfaces using their names. Packets can flow into, out of, or in both directions through both interfaces at any time, depending on how an application directs the packet flow.

Applications can also use the IX-API to retrieve information about an interface's properties, such as its speed or duplex mode, and to change the settings of those properties.

About the ODX Protocol

Because the network interfaces A and B are physically part of the Policy Accelerator, you cannot change their physical attributes. Therefore, it might be convenient to add an additional network interface that provides cable connector types that are different from those offered by the internal interfaces.

The ODX Protocol allows you to develop a logical interface between a NIC driver and a Policy Accelerator (PA-100) driver. You can use the ODX protocol for any variety of NIC that can communicate across the PCI bus to the Policy Accelerator.

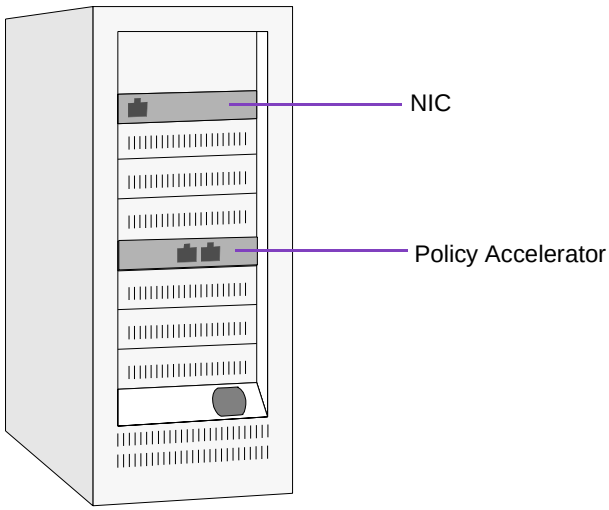
The ODX Protocol allows you to customize a driver for a network interface card (NIC) so that the NIC behaves as an interface named C for the Policy Accelerator. Any applications using the Policy Accelerator can then bind packet flow through interface C in the same way that they bind interfaces A and B.

Because each NIC is different, the ODX Protocol consists of two parts:

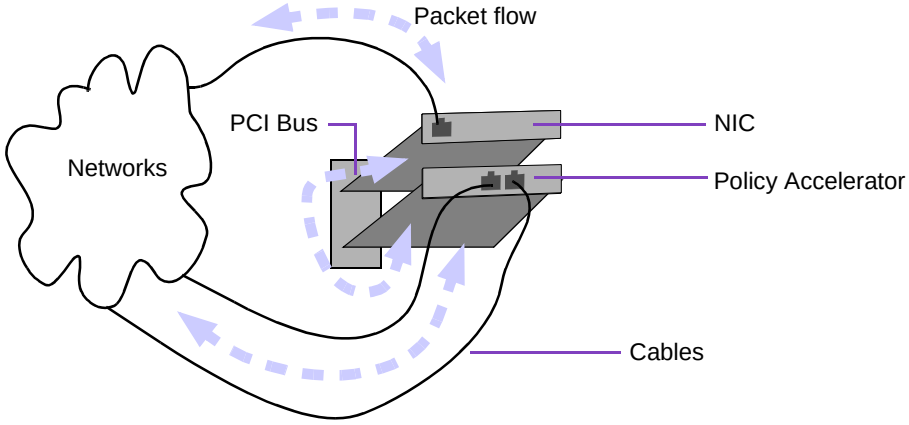
- PA-100 functions: These provide the NIC driver with information about the Policy Accelerator's packet buffers and provide some control over the Policy Accelerator's related behavior.
- NIC functions: These provide the PA-100 driver with access to packets flowing through the NIC and provide IXA applications with information about the NIC's status. You must implement these functions as part of your NIC driver; ODX specifies the format that these functions must take and the data types and structures that these functions use.

Connecting the NIC and the Policy Accelerator

The NIC and the Policy Accelerator communicate across the PCI bus to which they are both connected. No other physical connection is required, so they do not need to be seated near each other within the system, as shown in the following diagram:



In a system with a NIC driver customized using the ODX Protocol, you can have up to three physical connections to one or more networks—the A and B interfaces on the Policy Accelerator and the C interface on the NIC—as shown in one possible configuration in the following diagram:



One NIC for One Policy Accelerator

Currently, ODX supports only a one-to-one relationship between an external NIC and a Policy Accelerator:

- The NIC and the Policy Accelerator must be installed in the same host because they must be able to communicate through the host's PCI bus.
- Any specific NIC can communicate with only one Policy Accelerator at one time.
- Each Policy Accelerator can communicate with only one external NIC at one time.

What the ODX Protocol Includes

When you receive the ODX protocol, it includes the following:

- The `nbfif.h` header file; this provides the prototypes of functions that you must implement to make your NIC driver work with the Policy Accelerator
- The `nbpe.h` header file provides the prototypes of Policy Accelerator (PA-100) driver functions that your NIC driver will use as you implement the `nbfif.h` functions.

  The IX-API SDK libraries contain the code that implements the `nbpe.h` functions

- C-language source code for a sample (“reference”) implementation of the driver for a specific NIC
- `odxloop`, a diagnostic utility

How Applications Can Use the New Driver

After you have changed your NIC driver using ODX, IXA applications can bind Policy Accelerator packet flow through your NIC as interface C in the same way that they use the Policy Accelerator’s A and B interfaces.

If an application attempts to bind a C interface that does not exist, such as when a customized NIC is absent, an error occurs.

Chapter 2
Customizing a NIC Driver

This chapter explains how to use the Intel® Optimal Data Exchange (ODX) Protocol to customize your network interface card's (NIC) driver. After the customization, your NIC will behave as a network interface for a Policy Accelerator, receiving and transmitting packets on behalf of the Policy Accelerator.

This chapter contains the following sections:

- Prerequisites
- Customization Environment
- Implementing the ODX Functions
- Initiating Communication With the Policy Accelerator
- Terminating Communication With the Policy Accelerator
- Handling Packets
- Reporting and Evaluating Status
- Implementing the NIC's Properties Functions
- Implementing the NIC's Statistical Functions
- Installing and Testing the Driver

Prerequisites

NOTE: To do the customization described in this chapter, you must be familiar with your NIC driver, how it works, and how to modify it. See "Audience" on page vii for more about what you need to know.

To customize your NIC driver using the ODX protocol, you need:

- The Intel IX-API software developer's kit (SDK) installed on a host system running the Windows NT operating system
- An Intel Policy Accelerator board installed in the same host

- The ODX header files `nbfif.h` and `nbpe.h`, delivered as part of the IX-API SDK

To test the driver, you need:

- The diagnostic utility `odxloop`, located in *`NBinstallpath`*`/diagnostics`
- An *IXA application;* that is, one that uses the IX-API to move packets through interface C (your NIC) of the Policy Accelerator, such as the demonstration code

NOTE: The ODX protocol supports only applications that are written using the IX-API SDK for the Windows NT operating system. It does **not** currently support the IX-API SDK for other operating systems.

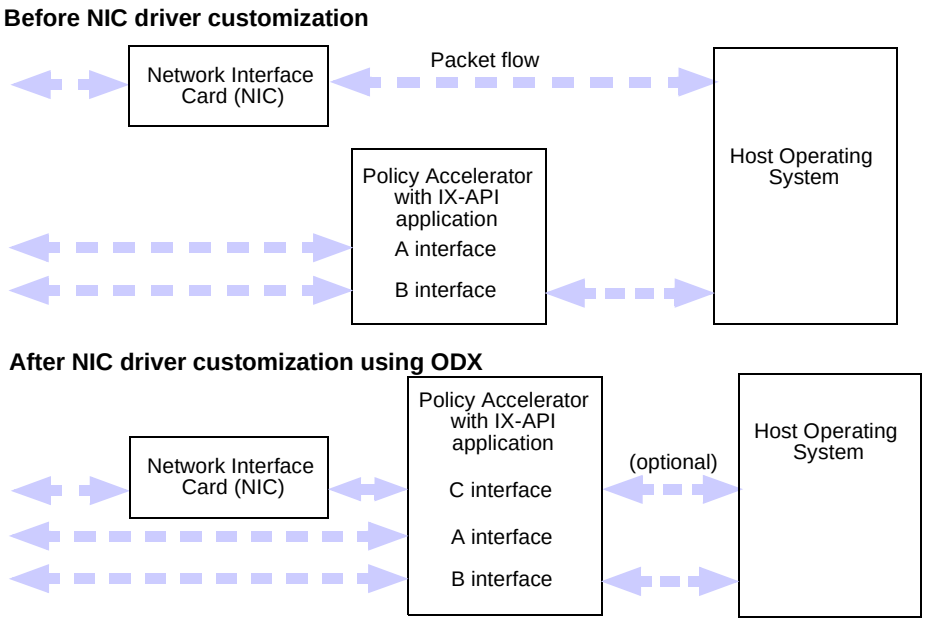
Customization Environment

The goal of customization is to change your NIC's driver so that packets going through the NIC automatically go through the Policy Accelerator, in such a way that the NIC acts as interface C for the Policy Accelerator. ODX provides the protocol for doing this.

This allows IXA applications to use the NIC for packet traffic.

The following diagram shows packets that normally flow through a NIC directly to the host for processing. After an ODX customization, the packets flow through the Policy Accelerator. The essential difference is that now applications using the IX-API SDK to handle packets on the Policy Accelerator can also handle packets coming through the NIC interface, and the packets might not need to be uploaded to the host at all.

This example shows an application that might direct some packets flowing through interface B to the host for additional processing.

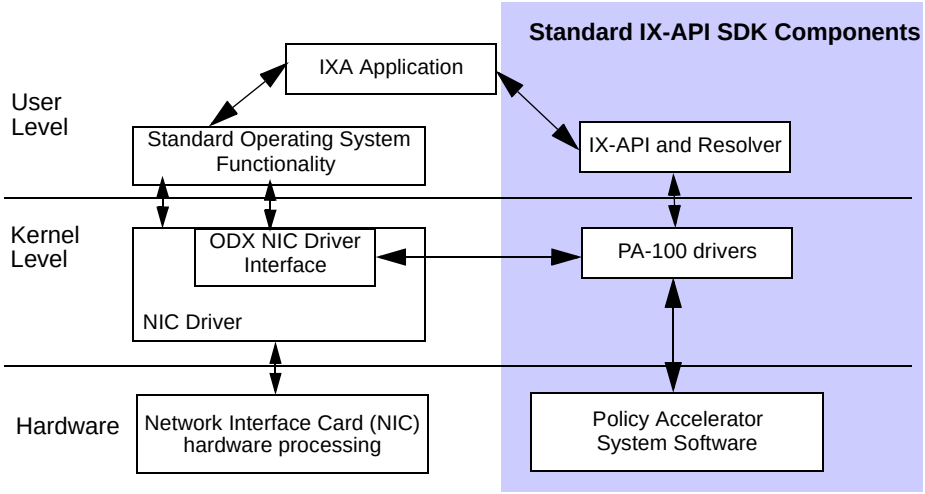


Software Components

The following diagram shows the components involved in processing packets:

- IXA application: Written by an application developer, it can use the IX-API to handle packets flowing into or out of the Policy Accelerator. It also can use all standard operating system (OS) functionality.
- IX-API and Resolver: Part of the IX-API SDK, these provide a software interface between an IXA application and the PA-100 drivers, which control the Policy Accelerator.
- NIC driver: This is your driver for your NIC.
- ODK NIC driver interface: This is your implementation of the functions described in the `nbfif.h` header file.
- PA-100 drivers: This includes an implementation of the functions described in the nbpe.h header file. This might be one or more drivers that control the operation of the Policy Accelerator and interact with the ODX NIC driver interface.
- NIC hardware processing: Typically this is packet management at the hardware level; for example, your NIC driver might identify buffer locations for packets, and the NIC hardware fills the buffers automatically.

- Policy Accelerator system software: Handles packets, buffers, memory, and such on the Policy Accelerator.



Implementing the ODX Functions

Because each NIC is different, the ODX Protocol consists of two sets of functions:

- PA-100 functions: These functions are provided as part of the IX-API SDK. They are described in the `nbpe.h` header file along with the necessary data types and structures that allow the NIC driver to get information about the Policy Accelerator's packet buffers and to have some control over the Policy Accelerator's related behavior.
- NIC functions: You must implement these functions as part of your NIC driver. They are described in the `nbfif.h` header file. They provide the PA-100 driver with access to packets flowing through the NIC and provide IXA applications with information about the NIC's status. The header file specifies the format that these functions must take and the data types and structures that these functions use, but you must provide the code to make them work.

CAUTION: Do not modify these header files.

Required and Optional Functions

The following table lists the functions descrbed in the `nbfif.h` header file. You must implement at least the required functions:

Used by PA-100 driver for:	Function name	Implementation notes
Initialization	`NBFIF_BindRecv` Function	Required.
Packet management	`NBFIF_BoundTxPacketsReady` Function	Optional. If not implemented, the PA-100 driver uses `NBFIF_PacketReady` instead.
	`NBFIF_PacketReady` Function	Required.
Property querying and setting	`NBFIF_GetPropCapability` Function	Optional, but required if either of the other two property functions are implemented.
	`NBFIF_GetProperties` Function	Optional
	`NBFIF_SetProperties` Function	Optional.
Statistical reporting	`NBFIF_ControlStatEngine` Function	Optional, but might be required for `NBFIF_GetStatistics` to operate.
	`NBFIF_GetStatCapability` Function	Optional, but required if `NBFIF_GetStatistics` is implemented.
	`NBFIF_GetStatistics` Function	Optional.
Resetting	`NBFIF_Reset` Function	Optional but recommended.
Termination	`NBFIF_UnbindRecv` Function	Required.

Basic Customization Steps

The basic steps, described in detail in the next sections, are as follows:

1. Modify your existing NIC driver code to do the following at the appropriate times:
   - Initiate communication with the PA-100 driver for the Policy Accelerator
   - Use the ODX functions described in `nbpe.h` to transmit and receive packets instead of using your normal code path
   - Terminate communication with the PA-100 driver

2. Implement the ODX functions described in `nbfif.h` and include them as part of your NIC driver. The PA-100 driver calls these functions as needed after your driver has initiated communication. These functions include the following actions:
   - Reset the NIC
   - Provide information about the NIC's properties and statistics to the PA-100 driver for delivery to a querying IXA application
   - Provide received packets to the Policy Accelerator
   - Get packets from the Policy Accelerator for transmission
3. Install and test the NIC driver.

Initiating Communication With the Policy Accelerator

Your NIC driver must initiate communication with the Policy Accelerator's PA-100 driver and provide a function for the PA-100 driver to use complete the connection.

Concepts

Your NIC driver initiates communication by using a PA-100 version of the Windows NT IOCTL command. As part of this initialization, you pass the PA-100 driver:

- A unique handle
- Pointers to the ODX functions that you have implemented

When the PA-100 driver receives your initialization request, it calls a binding function, defined as part of ODX, that you have implemented. It uses this call to pass to your NIC driver:

- A unique handle
- Pointers to the ODX functions in the PA-100 driver

✓ **NOTE:** After communication is established, the NIC can function as an interface only for the Policy Accelerator, not for the host system as a stand-alone NIC.

Implementing

✓ **NOTE:** A recommended technique is to create functions like *yourdriver*`_OpenIXDevice` and *yourdriver*`_CloseIXDevice` inside your NIC driver to perform the initialization and termination functions. This encapsulates the following steps, so that if the ODX protocol

changes or your driver is eventually ported to another operating system, you can change the details without changing the primary function name.

Your NIC driver must:

1. Initiate a connection to the PA-100 driver and register its functions with the PA-100 driver. To do this, use the ODX version of the standard IOCTL call, `IOCTL_NBOOST_FOREIGN_REGISTER`, by passing the `_NBFIF_REGISTER_PARAM` structure, which includes the following:
   - A unique handle
   - Pointers to your implemented functions
   - Driver identifiers

NOTE: Some functions are optional. If you choose not to implement a function, you must pass a NULL in place of the function pointer to `IOCTL_-NBOOST_FOREIGN_REGISTER`.

2. Include an implementation of the `NBFIF_BindRecv` function.

   The PA-100 driver calls this function after it receives your IOCTL call and passes it the following:
   - A unique handle, which the NIC driver will use in future calls to PA-100 driver functions
   - Pointers to its functions

   The NIC driver must:

   a. Save these function pointers.

   b. Return a status based on the success of this and any operations other operations performed here.

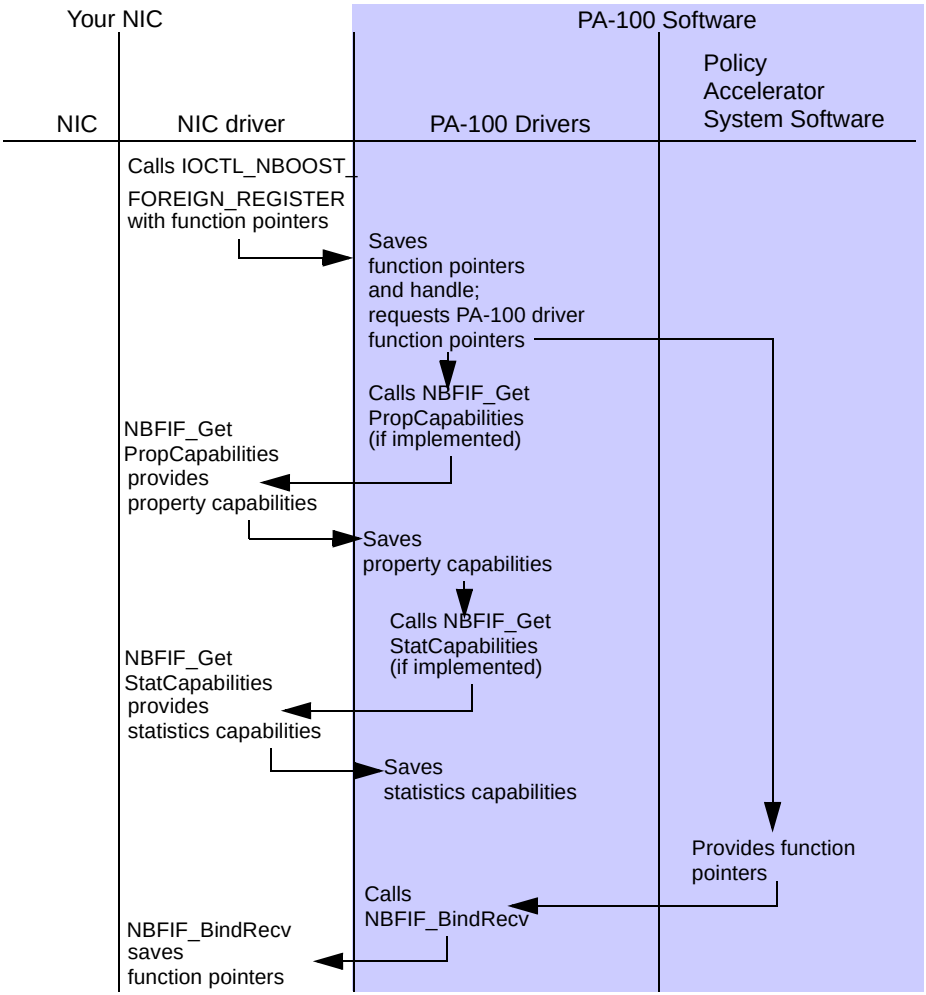
NOTE: You might want to include Step 3 as part of `NBFIF_BindRecv`, but you can do it elsewhere if desired.

3. Initialize packet buffers and initiate packet flow as described in “Initiating Packet Flow” on page 18.
4. In addition, the PA-100 driver looks for the following functions during initialization and uses them if they are implemented:
   - `NBFIF_GetPropCapability` function as described in “Implementing the NIC’s Properties Functions” on page 23
   - `NBFIF_GetStatCapability` function as described in “Implementing the NIC’s Statistical Functions” on page 25

Initialization Process

The following diagram shows what happens when your NIC driver calls `IOCTL_NBOOST_FOREIGN_REGISTER` to initiate communcation:

Initialization



For More Information

- “IOCTL_NBOOST_FOREIGN_REGISTER Function” on page 83
- “NBFIF_BindRecv Function” on page 49
- “Handling Packets” on page 17
- “Reporting and Evaluating Status” on page 23

- "Implementing the NIC's Properties Functions" on page 23
- "Implementing the NIC's Statistical Functions" on page 25

Terminating Communication With the Policy Accelerator

Your NIC driver is also responsible for terminating communication with the Policy Accelerator's PA-100 driver and for providing a function for the PA-100 driver to use to complete the termination.

Concepts

Your NIC driver terminates communication by using a PA-100 version of the Windows NT IOCTL command. As part of this termination, you pass the PA-100 driver:

- The unique handle that the NIC driver originally passed to the PA-100 driver
- The unique handle that the PA-100 driver originally passed to the NIC driver

When the PA-100 driver receives your termination request, it calls an unbinding function, defined as part of ODX, that you have implemented and in which you terminate all activity.

Implementing

Your NIC driver must:

1. Terminate the connection to the PA-100 driver and unregister its functions with the PA-100 driver. To do this, use the ODX version of the standard IOCTL call, `IOCTL_NBOOST_FOREIGN_UNREGISTER`, by passing the `_NBFIF_UNREGISTER_PARAM` structure, which includes the following:
   - The unique handle that the NIC driver originally passed to the PA-100 driver
   - The unique handle that the PA-100 driver originally passed to the NIC driver
2. Include an implementation of the `NBFIF_UnbindRecv` function.

   The PA-100 driver calls this function after it receives your IOCTL call and passes it the following:
   - A unique handle

   This function must cleanly terminate its connection to the Policy Accelerator. Recommended steps include:

   a. Stop sending packets to the Policy Accelerator.

   b. Free all buffers in the NIC, if any exist.

c. Return all allocated buffers to the Policy Accelerator using `NBPE_ReturnXmitPEBuffers`.

d. NULL the PA-100 driver function pointers that the NIC driver's `NBFIF_BindRecv` function saved.

   This ensures that the PA-100 driver does not accidentally attempt to use functions that are now invalid.

e. Stop the NIC hardware

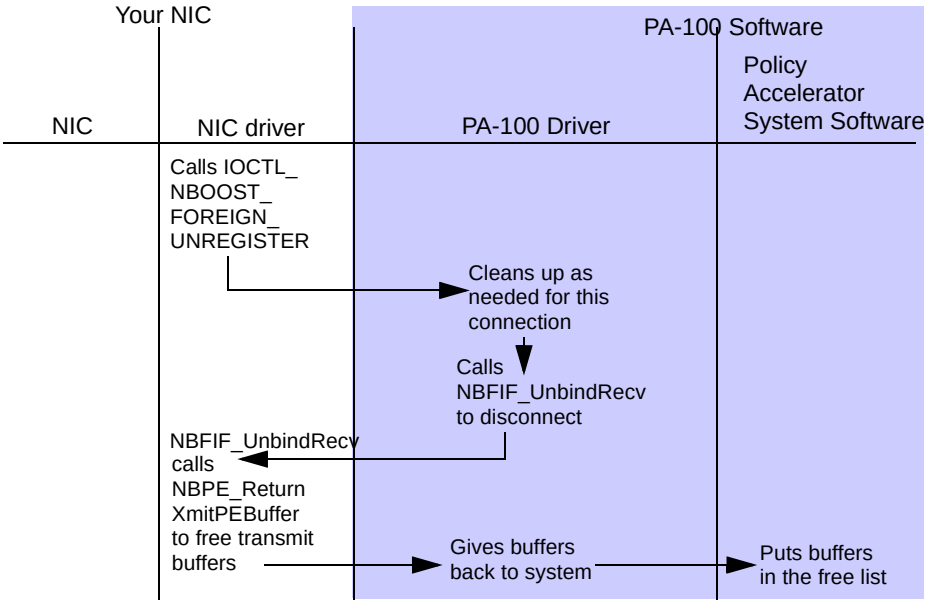
f. Return a status based on the success of these operations.

NOTE: After the `IOCTL_NBOOST_FOREIGN_UNREGISTER` function completes, the NIC driver can no longer call any of the PA-100 driver functions.

Termination Process

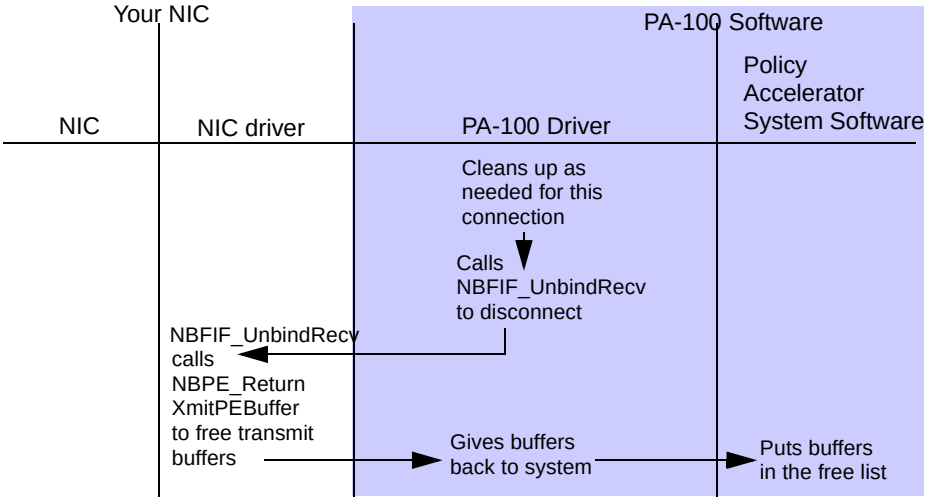
The following diagram shows what happens when your NIC driver calls `IOCTL_NBOOST_FOREIGN_UNREGISTER` to terminate communcation:

Termination



It is also possible that the PA-100 driver will initiate termination by calling `NBFIF_UnbindRecv`. In this case, the process is as follows:

Termination



For More Information

- “IOCTL_NBOOST_FOREIGN_UNREGISTER Function” on page 84
- “NBFIF_UnbindRecv Function” on page 65

Handling Packets

The goal of customizing the NIC driver using ODX is to have the NIC driver receive and transmit packets on behalf of the Policy Accelerator, so that the NIC acts like a network interface to the Policy Accelerator.

Concepts

For both transmitting and receiving packets, the PA-100 driver provides the NIC driver with pointers to packet buffers on the Policy Accelerator. The NIC driver places received packets into these buffers instead of into its normal receive buffers, and it takes packets to be transmitted from these buffers instead of from its normal transmit buffers.

The PA-100 driver provides the packet buffers as rings of buffers, with separate transmit and receive rings. For each ring, the PA-100 driver provides a pointer to the beginning of the currently available buffers in the ring, to end of the currently available buffers the ring, and a mask to apply to the pointers to accommodate the ongoing wrap of buffers in the ring.

When the NIC driver has received packets and placed them into the receive ring for the Policy Accelerator to process, it notifies the PA-100 driver that packets are ready. In reverse, when the Policy Accelerator has placed packets for transmission into the transmit ring, the PA-100 driver notifies the NIC driver that packets are ready.

You implement in the NIC driver:

- "NBFIF_PacketReady Function" on page 62
- "NBFIF_BoundTxPacketsReady Function" on page 51 (optional)

Your NIC driver uses the following functions

- "NBPE_GetBufferTypes Function" on page 85
- "NBPE_GetRecvPEBufRing Function" on page 86
- "NBPE_GetTransmitPEBufRing Function" on page 87
- "NBPE_PacketReady Function" on page 88
- "NBPE_ReturnXmitPEBuffer Function" on page 90

Initiating Packet Flow

The NIC driver should set up for transmit and receive packet flow as part of the implementation of the `NBFIF_BindRecv` function, described in "Initiating Communication With the Policy Accelerator" on page 12.

1. Verify that the NIC hardware is halted and, if not, halt it.

   This ensures that packets are not flowing through the NIC while the buffers are set up.

2. Call `NBPE_GetBufferTypes` to get a list of buffer types that the Policy Accelerator supports.
3. Select the buffer type that is appropriate for your NIC.
4. Pass the buffer type to `NBPE_GetRecvPEBufRing` to get the location of the Policy Accelerator's empty packet buffer receive ring.

   This function returns:

   - A pointer to the beginning (base) buffer in the receive ring
   - An index mask to account for wrapping around the ring
   - A produce index pointer, which points to the end of the currently available buffers in the receive ring

5. Set a consume-buffer pointer to the base pointer.
6. Use this location as the NIC's receive buffer; that is, any packet received by the NIC should go into this receive ring.

7. Do any initialization required to start the NIC receiving packets into this ring.
8. Pass the same buffer type to `NBPE_GetTransmitPEBufRing` to get the location of the Policy Accelerator's empty packet buffer transmit ring.

   This function returns:

   - A pointer to the beginning (base) buffer in the transmit ring
   - An index mask to account for wrapping around the ring
   - A produce index pointer, which points to the end of the currently available buffers in the transmit ring
9. Set a separate consume-buffer pointer to the base pointer.
10. Start the NIC hardware.
11. If all initialization has been successful, packets begin flowing.

Receiving Packets

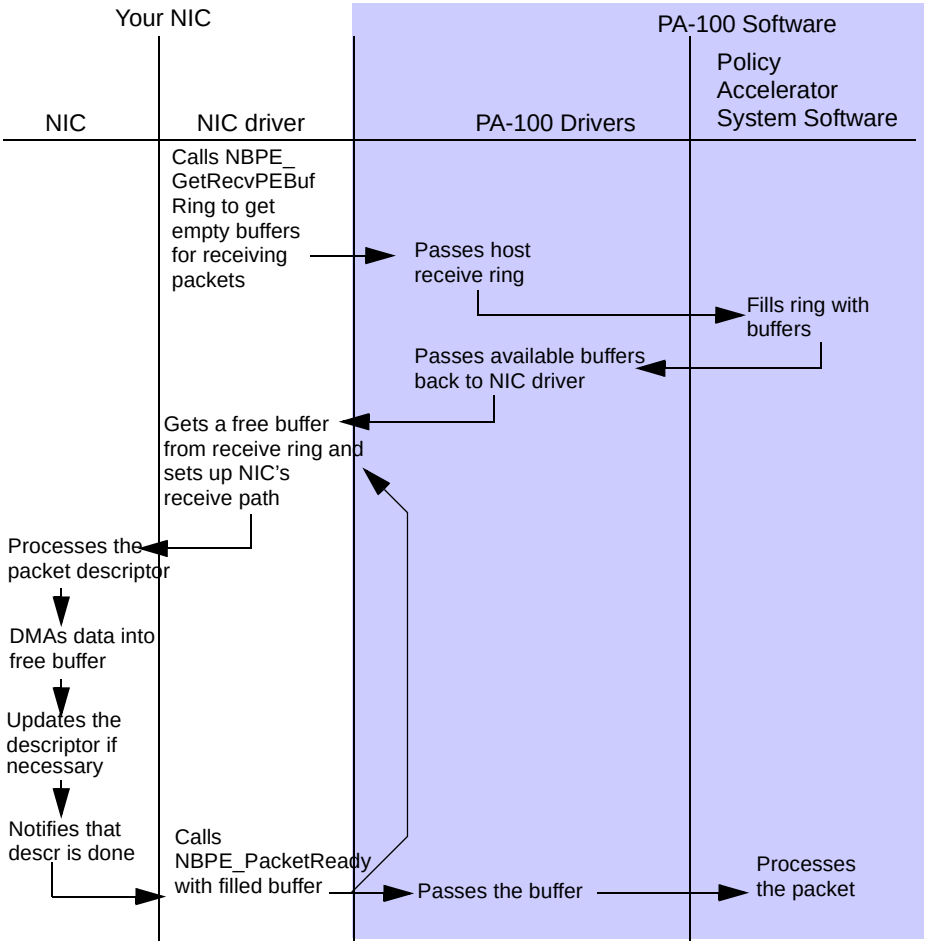
For each packet that the NIC receives and places into a buffer in the receive ring, the NIC driver should do the following:

1. Notify the Policy Accelerator that a packet is ready by calling `NBPE_PacketReady`.
2. Increment its received consume-buffer pointer by 1.
3. Apply the receive index mask to the consume-buffer pointer and compare it to the value pointed to by the receive produce index.

   If they are equal, then there is currently no more space in the receive buffer ring and the NIC driver must decide how to handle additional incoming packets, typically by dropping them.

Receive Diagram

The following diagram shows the initialization and processing of packets received by the NIC on behalf of the Policy Accelerator.



Transmitting Packets

For each packet that the Policy Accelerator prepares for transmittal by placing it into a buffer in the transmit ring, the following occurs:

1. The PA-100 driver notifies the NIC driver that a packet is ready by calling `NBFIF_BoundTxPacketsReady`.
2. In the `NBFIF_BoundTxPacketsReady` function, the NIC driver should do the following:
   a. Transmit the packet
   b. Increment its transmitted consume-buffer pointer by 1.

c. Apply the transmit index mask to the consume-buffer pointer and compare it to the value pointed to by the transmit produce index.

   If they are equal, then there are currently no more packets in the transmit buffer ring; otherwise, continue with Step a. for the next packet.

Alternative Transmit Method

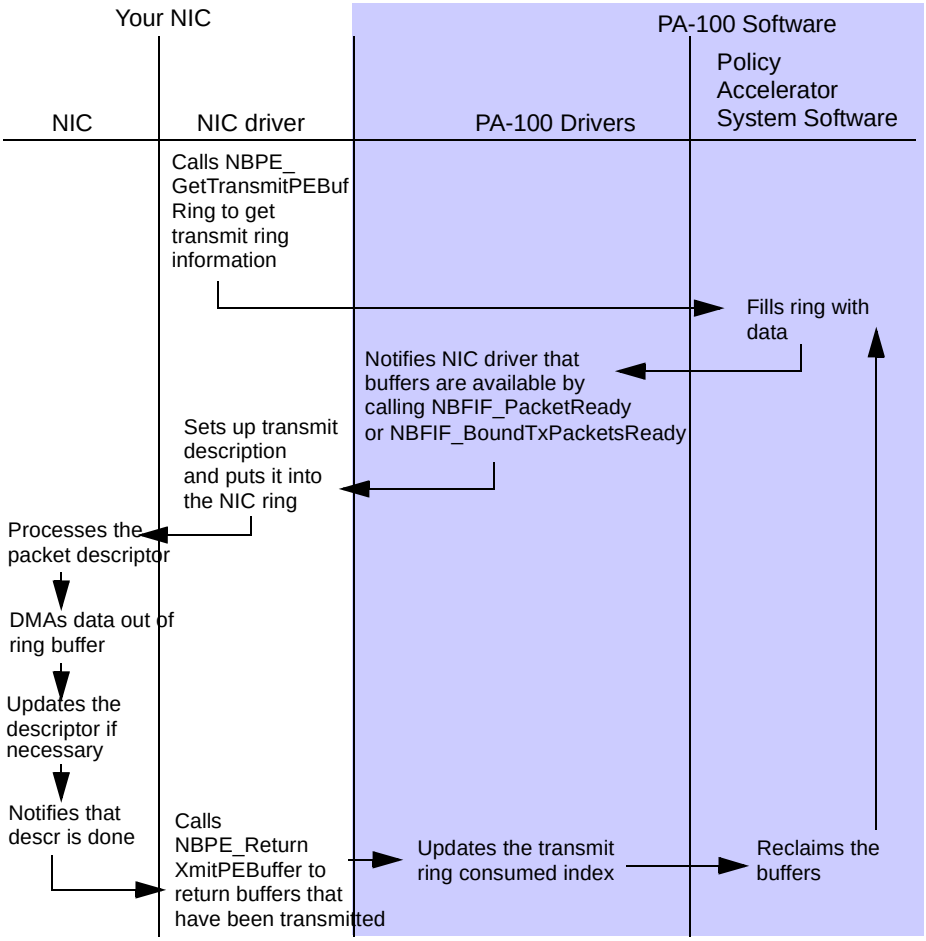
The preceding method is the fastest method for transmitting packets. If, however, you want to transmit packets one at a time rather than from a ring, such as for testing, you can do so as follows:

1. Do not call `NBPE_GetTransmitPEBufRing` during initialization.
2. Do not implement `NBFIF_BoundTxPacketsReady`; instead, pass a NULL for this function's pointer to the PA-100 driver in the `IOCTL_NBOOST_-FOREIGN_REGISTER` function.
3. Implement `NBFIF_PacketReady`.

   Whenever `NBFIF_BoundTxPacketsReady` is not implemented, the PA-100 driver calls `NBFIF_PacketReady` each time a packet is ready for transmit. It passes a pointer to the packet and the length of the packet; the NIC driver simply transmits this packet from this location.

Transmit Diagram

The following diagram shows initialization and packets flowing from the Policy Accelerator to the NIC:



For More Information

- “NBFIF_PacketReady Function” on page 62
- “NBFIF_BoundTxPacketsReady Function” on page 51 (optional)
- “NBPE_GetBufferTypes Function” on page 85
- “NBPE_GetRecvPEBufRing Function” on page 86
- “NBPE_GetTransmitPEBufRing Function” on page 87
- “NBPE_PacketReady Function” on page 88
- “NBPE_ReturnXmitPEBuffer Function” on page 90

Reporting and Evaluating Status

This section describes how you return status to the PA-100 driver and how the PA-100 driver returns status to you.

Concepts

Each of the functions that you implement must return a status code indicating whether the function succeeded. The header file `nbfif.h` defines a UINT32 type named `NBFIF_STATUS` to assist with this.

NOTE: The `NBFIF_STATUS` type conforms to the Windows NT operating system's standard 32-bit error code format, as summarized in "NBFIF_STATUS Type" on page 44. Refer to your operating system documentation for details.

The header file also defines a small set of useful 32-bit statuses that you can return.

NOTE: The PA-100 driver evaluates your returned status only for success or failure; it does not take any actions based on any particular failure, but rather simply passes the status up to the calling application if applicable.

Interpreting Status from the PA-100 driver

Each of the PA-100 driver functions returns a status of type NBPE_STATUS.

Resetting Devices

Each driver has the option of requesting that the other driver's device be reset into a known, clean state. This might be appropriate when errors of unknown origin continue to occur. The functions are:

- "NBFIF_Reset Function" on page 63: Implement this so that the PA-100 driver can call it to set the NIC into a known, clean state.
- "NBPE_Reset Function" on page 89: Call it to set the Policy Accelerator into a known, clean state.

Implementing the NIC's Properties Functions

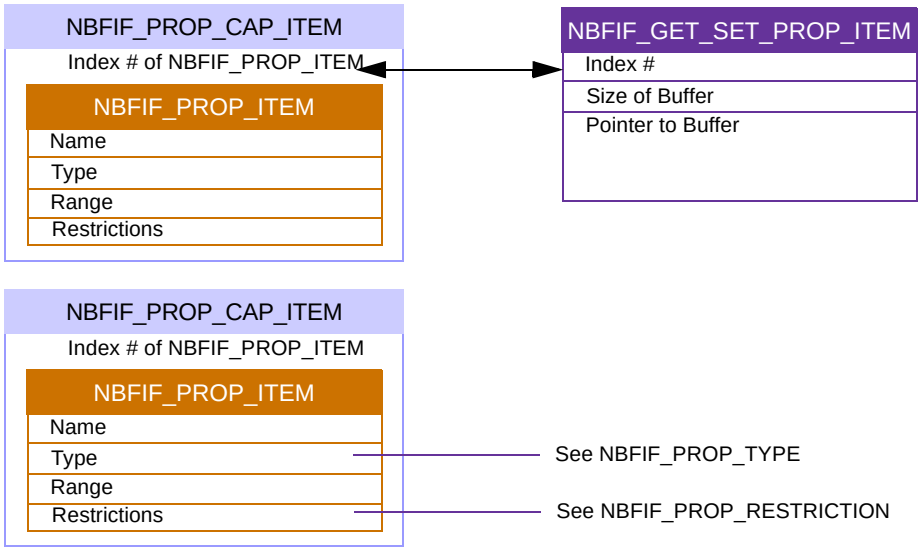
If you want applications to be able to query or set properties of your NIC—such as its speed or its duplex mode—you can implement the property functions. These functions allow the PA-100 driver to respond to an application that uses the IX-API SDK to:

- Request a list of available properties (PA-100 driver calls the `NBFIF_GetPropCapability` Function)
- Request the current settings of properties (PA-100 driver calls the `NBFIF_GetProperties` Function)
- Change the settings of properties (PA-100 driver calls the `NBFIF_SetProperties` Function)

These functions are optional; however, you must implement `NBFIF_GetPropCapability` if you implement either of the other functions.

You define and access a NIC's properties using the following structures:

Structure	Description
`_NBFIF_GET_SET_PROP_ITEM` Structure	Holds the value of a property.
`_NBFIF_PROP_ITEM` Structure	Describes a property.
`_NBFIF_PROP_CAP_ITEM` Structure	Associates a property description with an index number as part of a properties list.



For More Information

- "NBFIF_GetPropCapability Function" on page 54
- "NBFIF_GetProperties Function" on page 56
- "NBFIF_SetProperties Function" on page 64

Implementing the NIC's Statistical Functions

If you want applications to be able to query or reset statistical information about your NIC—such as the quantity of transmit or receive errors—you can implement the statistics functions. These functions allow the PA-100 driver to respond to an application that uses the IX-API SDK to:

- Request a list of available statistics (PA-100 driver calls the `NBFIF_GetStatCapability` Function)
- Request the current values of statistics (PA-100 driver calls the `NBFIF_GetStatistics` Function)
- Turn the NIC driver's statistical engine on or off (PA-100 driver calls the `NBFIF_ControlStatEngine` Function)

These functions are optional; however, you must implement `NBFIF_GetStatCapability`—and possibly `NBFIF_ControlStatEngine`, if your NIC driver allows its caller to turn the statistics calculations on and off—to implement `NBFIF_GetStatistics`.

For More Information

- Optionally, "NBFIF_GetStatCapability Function" on page 58
- Optionally, "NBFIF_GetStatistics Function" on page 60

Installing and Testing the Driver

CAUTION: Ensure that the NIC driver doesn't start until the Resolver is started, or unexpected results can occur.

Installation/ Configuration

The PA-100 driver is loaded automatically by the SDK when the host boots.

For the Resolver to successfully recognize the existence of an ODX NIC driver and to start the NIC driver, you must:

- Set the registry

Testing

To test the customized NIC driver after installation:

1. Run the ODX test set-up utility, `odxloop`, in a command shell on the host system that contains the Policy Accelerator and its companion NIC.

   This tool configures the Policy Accelerator to reflect packets back to the NIC for testing purposes only. The tool is in the *`NBInstallPath`*`/diagnostics` directory. See the *IX-API SDK Reference* chapter on command-line tools for details.

2. Send packets to the NIC.
3. Verify that all of the packets are returned to the NIC from the Policy Accelerator in the same order and state.
4. Reboot the host to return the Policy Accelerator to its default configuration.

Sample Application

The product installation includes a sample IX-API SDK application that uses the C interface made possibly by your ODX NIC driver. The demo is in the following location:

`NBInstallPath``/demo/ODXFilter`

Chapter 3
Alphabetic Reference: NBFIF (NIC Driver) Functions, Types, and Structures

This chapter contains an alphabetic reference to the Intel® Optimal Data Exchange (ODX) Protocol functions that you must implement so that the PA-100 driver can call them, and to the types and structures that you use to assist with the implementation.

It describes each function and data element, and explains in detail how to implement each function.

It contains the following sections:

Overview

The `nbfif.h` header file provides the prototypes of NIC driver functions that you must implement and that the PA-100 driver will call. Your driver code must include this file.

CAUTION: Do not modify this header file.

NBFIF Types and Structures Alphabetic Reference

NBFIF Types and Structures Summary

The nbfif.h header file contains the following data types and structures that the NIC driver and the PA-100 driver use to pass information between themselves:

Data Element	Description
_NBFIF_DEV_HANDLE Type	A unique handle that the NIC driver must provide to the PA-100 driver.
_NBFIF_GET_SET_PROP_ITEM Structure	Provides an indexed structure that the PA-100 driver and the NIC driver can use to exchange information about the settings of the NIC's properties.
_NBFIF_GET_STAT_ITEM Structure	Provides an indexed structure that the PA-100 driver and the NIC driver can use to exchange information about the NIC's statistical values.
_NBFIF_PROP_CAP_ITEM Structure	Describes properties supported by your NIC and associates an index number with each property.
_NBFIF_PROP_ITEM Structure	Describes, for a property supported in the NIC driver, the property's name, its possible values, and whether the property can be viewed or changed.
_NBFIF_PROP_RESTRICTION Enumeration	Describes whether applications can view a NIC property's current setting, change it, or both.
_NBFIF_PROP_TYPE Enumeration	Describes the data type of the setting value of a NIC property.
_NBFIF_REGISTER_PARAM Structure	Description of the NIC's ODX interface and identifiers for the PA-100 drivers for use in registering the NIC driver with the PA-100 driver.

Data Element	Description
_NBFIF_STAT_CAP_ITEM Structure	Describes statistical items supported by your NIC and associates an index number with each item.
_NBFIF_STAT_CNTRL Enumeration	Provides valid settings for turning the NIC's statistical calculations on and off.
NBFIF_STATUS Type	A status code that each of your functions must return, indicating the success or failure of the function.
_NBFIF_UNREGISTER_PARAM Structure	The NIC driver's description of its ODX interface for use in unregistering itself from the PA-100 driver.

_NBFIF_DEV_HANDLE Type

A unique handle that the NIC driver must provide to the PA-100 driver.

```
typedef PVOID NBFIF_DEV_HANDLE;
```

Description

The PA-100 driver passes a pointer of this type to each NIC driver function call that it uses.

You can define the actual content of this type in any manner that you find useful. For example, you could include state information for the NIC driver.

This identifies the PA-100 driver to the NIC driver of which it is making a request. Therefore, you must define the pointer before you call the PA-100 driver's `IOCTL_NBOOST_FOREIGN_REGISTER` Function, because this function calls the `NBFIF_BindRecv` Function, which initializes the handle.

See Also

- "IOCTL_NBOOST_FOREIGN_REGISTER Function" on page 83

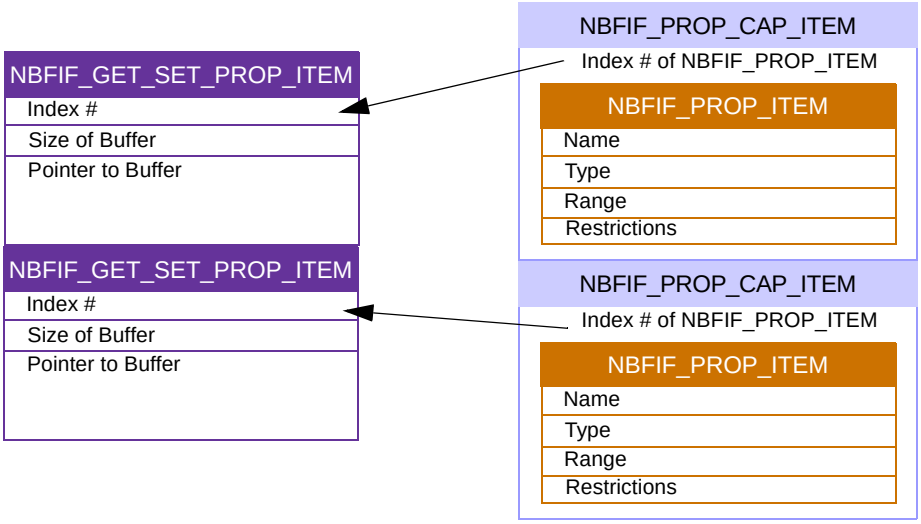
_NBFIF_GET_SET_PROP_ITEM Structure

Provides an indexed structure that the PA-100 driver and the NIC driver can use to exchange information about the settings of the NIC's properties.

```
typedef struct _NBFIF_GET_SET_PROP_ITEM {
    UINT32 propIndx;
    UINT32 bufSizeInBytes;
    PVOID  pValueBuf;
          } NBFIF_GET_SET_PROP_ITEM, *PNBFIF_GET_SET_PROP_ITEM;
```

Element	Definition
propIndx	The index number of the desired property within the table of properties. The PA-100 driver specifies this index number from the valid property indexes that the NIC driver provided in NBFIF_PROP_CAP_ITEM.
bufSizeInBytes	The size of the pValueBuf buffer, specified by the PA-100 driver.
pValueBuf	A buffer provided by the PA-100 driver to contain the value of a property. When the PA-100 driver calls NBFIF_GetProperties, the NIC driver must place the value of the specified property into this buffer. When the PA-100 driver calls NBFIF_SetProperties, it places the desired value into this buffer and the NIC driver must use it to set the specified NIC property.

Description



See Also

- “Implementing the NIC’s Properties Functions” on page 23
- “NBFIF_GetProperties Function” on page 56
- “NBFIF_SetProperties Function” on page 64
- “_NBFIF_PROP_CAP_ITEM Structure” on page 34
- “_NBFIF_PROP_ITEM Structure” on page 37

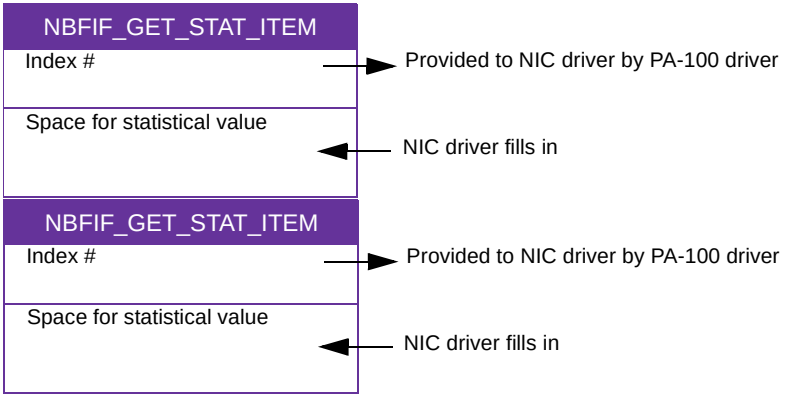
_NBFIF_GET_STAT_ITEM Structure

Provides an indexed structure that the PA-100 driver and the NIC driver can use to exchange information about the NIC's statistical values.

```
typedef struct _NBFIF_GET_STAT_ITEM {
     UINT32 statIndx;
     UINT32 statValue;
            } NBFIF_GET_STAT_ITEM, *PNBFIF_GET_STAT_ITEM;
```

Element	Definition
statIndx	The index number of the statistic within the table of statistics. The PA-100 driver specifies this index number from the valid property indexes that the NIC driver provided in NBFIF_PROP_CAP_ITEM.
statValue	Value of an individual statistic.

Description



See Also

- "Implementing the NIC's Statistical Functions" on page 25
- "NBFIF_GetStatistics Function" on page 60

_NBFIF_PROP_CAP_ITEM Structure

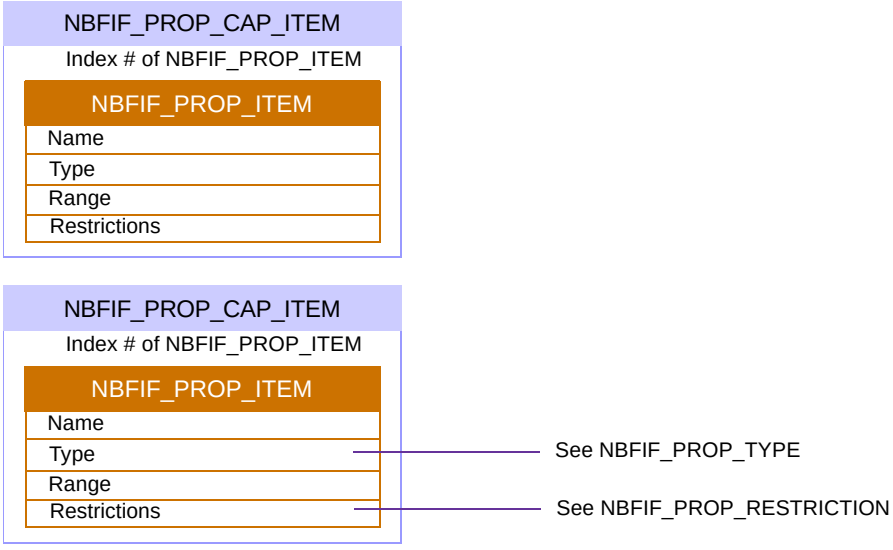
Describes properties supported by your NIC and associates an index number with each property.

```
typedef struct _NBFIF_PROP_CAP_ITEM {
     UINT32          propIndx; /* corresponding index */
     NBFIF_PROP_ITEM propItem;
            } NBFIF_PROP_CAP_ITEM, *PNBFIF_PROP_CAP_ITEM;
```

Element	Definition
propIndx	The index number of the property within the table of properties. The NIC driver can use any arbitrary numbering scheme for this index. For your convenience, ODX includes some optional predefined index values.
propItem	Information about an individual property. See "_NBFIF_PROP_ITEM Structure" on page 37.

Description

You implement NBFIF_GetPropCapability to fill in this table to describe the properties supported by your NIC. For example, this structure looks as follows for a NIC with two properties:



Predefined Index Values

You can use the following C-language defines as index numbers for the properties, or you can use your own index numbers. The PA-100 driver does not evaluate or use the index numbers in any manner; it simply passes the information up to the calling application if applicable. These defines appear in `nbfif.h` as in the following example:

```
#define NBFIF_PROP_GET_LINK_SPEED      0xFE000000L
```

NOTE: For your convenience in the Windows NT operating system, these property indexes map directly to NDIS defined object identifiers. You can use these index values, or not, as appropriate.

Property Index Predefined Values	Definition
NBFIF_PROP_GET_LINK_SPEED	0xFE000000L
NBFIF_PROP_SET_LINK_SPEED	0xFE000001L
NBFIF_PROP_GET_DUPLEX_MODE	0xFE000002L
NBFIF_PROP_SET_DUPLEX_MODE	0xFE000003L
NBFIF_PROP_GET_LINK_STATUS	0xFE000004L
NBFIF_PROP_ENABLE_INTERFACE	0xFE000005L
NBFIF_PROP_DISABLE_INTERFACE	0xFE000006L
NBFIF_PROP_ENABLE_UNICAST_PROMISCUOUS_OPS	0xFE000007L
NBFIF_PROP_DISABLE_UNICAST_PROMISCUOUS_OPS	0xFE000008L
NBFIF_PROP_ENABLE_MULTICAST_PROMISCUOUS_OPS	0xFE000009L
NBFIF_PROP_DISABLE_MULTICAST_PROMISCUOUS_OPS	0xFE000010L
NBFIF_PROP_GET_UNICAST_MAC_ADDR	0xFE000011L
NBFIF_PROP_SET_UNICAST_MAC_ADDR	0xFE000012L
NBFIF_PROP_GET_MULTICAST_FILTER_LIST	0xFE000013L
NBFIF_PROP_ADD_MULTICAST_ADDR	0xFE000014L
NBFIF_PROP_DEL_MULTICAST_ADDR	0xFE000015L

Property Index Predefined Values	Definition
NBFIF_PROP_GET_MAX_FRAME_SIZE	0xFE000016L
NBFIF_PROP_SET_MAX_FRAME_SIZE	0xFE000017L
NBFIF_PROP_GET_INTERFACE_TYPE	0xFE000018L
NBFIF_PROP_GET_MEDIA_TYPE	0xFE000019L

See Also

- “Implementing the NIC’s Properties Functions” on page 23
- “NBFIF_GetPropCapability Function” on page 54
- “_NBFIF_GET_SET_PROP_ITEM Structure” on page 31
- “_NBFIF_PROP_ITEM Structure” on page 37
- “_NBFIF_PROP_TYPE Enumeration” on page 40
- “_NBFIF_PROP_RESTRICTION Enumeration” on page 39

_NBFIF_PROP_ITEM Structure

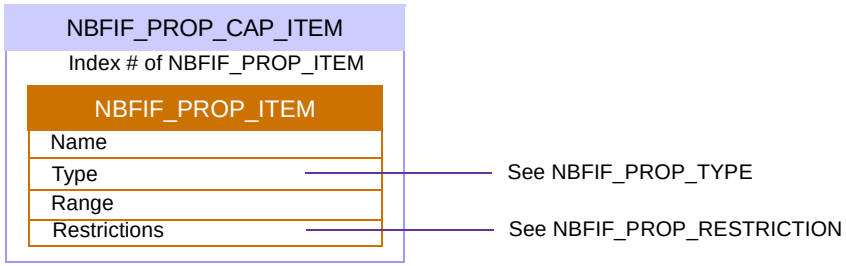
Describes, for a property supported in the NIC driver, the property's name, its possible values, and whether the property can be viewed or changed.

```
typedef struct _NBFIF_PROP_ITEM {
     CHAR                    propName [NBFIF_MAX_NAME];
     NBFIF_PROP_TYPE         propType;
     INT32                   range;
     NBFIF_PROP_RESTRICTION  restriction;
              } NBFIF_PROP_ITEM, *PNBFIF_PROP_ITEM;
```

Element	Definition
propName	A name of your choice that describes a property.
propType	Describes the data type of the property's setting. See "_NBFIF_PROP_TYPE Enumeration" on page 40.
range	The upper limit of the range of valid values for this property. A value of -1 means that this property has no upper limit, or an upper limit does not apply. Otherwise, this element's meaning depends on *propType* as follows: ■ Integer and Integer Array: The property's maximum valid value, where zero (0) is always the minimum valid value. For an array, this applies to each element of the array. ■ Ether_addr: The maximum quantity of Ethernet addresses allowed for this property. ■ Str_list: The maximum number of strings allowed in the list that defines this property. ■ Boolean: Not relevant. Ignored if provided. ■ Mask: Not relevant. Ignored if provided.
restriction	Describes whether applications can view the property's current setting, change it, or both. See "_NBFIF_PROP_RESTRICTION Enumeration" on page 39.

Description

The _NBFIF_PROP_CAP_ITEM Structure associates one or more of these structures with unique indexes for use by the NBFIF_GetPropCapability Function.



See Also

- “Implementing the NIC’s Properties Functions” on page 23
- “NBFIF_GetPropCapability Function” on page 54
- “_NBFIF_PROP_CAP_ITEM Structure” on page 34
- “_NBFIF_PROP_RESTRICTION Enumeration” on page 39
- “_NBFIF_PROP_TYPE Enumeration” on page 40

_NBFIF_PROP_RESTRICTION Enumeration

Describes whether applications can view a NIC property's current setting, change it, or both.

```
typedef enum _NBFIF_PROP_RESTRICTION {
     NBFIF_RESTR_READ_ONLY,
     NBFIF_RESTR_WRITE_ONLY,
     NBFIF_RESTR_READ_WRITE
            } NBFIF_PROP_RESTRICTION;
```

Element	Definition
NBFIF_RESTR_READ_ONLY	The NIC driver allows applications to view a specified property's current setting, but not to change it.
NBFIF_RESTR_WRITE_ONLY	The NIC driver allows applications to change the property's setting, but not to view its current setting.
NBFIF_RESTR_READ_WRITE	The NIC driver allows applications to view and to change the property's setting.

See Also

- "Implementing the NIC's Properties Functions" on page 23
- "_NBFIF_PROP_ITEM Structure" on page 37

_NBFIF_PROP_TYPE Enumeration

Describes the data type of the setting value of a NIC property.

```
typedef enum _NBFIF_PROP_TYPE {
     NBFIF_PROP_INTEGER,
     NBFIF_PROP_INTEGER_ARRAY,
     NBFIF_PROP_ETHER_ADDR,
     NBFIF_PROP_STR_LIST,
     NBFIF_PROP_MASK,
     NBFIF_PROP_BOOLEAN
            } NBFIF_PROP_TYPE;
```

Element	Definition
NBFIF_PROP_INTEGER	
NBFIF_PROP_INTEGER_ARRAY	
NBFIF_PROP_ETHER_ADDR	
NBFIF_PROP_STR_LIST	
NBFIF_PROP_MASK	
NBFIF_PROP_BOOLEAN	

Description

See Also

- “Implementing the NIC’s Properties Functions” on page 23
- “_NBFIF_PROP_ITEM Structure” on page 37

_NBFIF_REGISTER_PARAM Structure

Description of the NIC's ODX interface and identifiers for the PA-100 drivers for use in registering the NIC driver with the PA-100 driver.

```
typedef struct _NBFIF_REGISTER_PARAM {
    NBFIF_DEV_HANDLE             hDev;
    UINT32                       peId;
    UINT32                       ifId;
         /* Actual function pointers */
    NBFIF_RESET                  NBFIF_Reset;
    NBFIF_BIND_RECV              NBFIF_BindRecv;
    NBFIF_UNBIND_RECV            NBFIF_UnbindRecv;
    NBFIF_BOUND_TX_PACKETS_READY NBFIF_BoundTxPacketsReady;
    NBFIF_PACKET_READY           NBFIF_PacketReady;
    NBFIF_GET_PROP_CAP           NBFIF_GetPropCapability;
    NBFIF_GET_PROP               NBFIF_GetProperties;
    NBFIF_SET_PROP               NBFIF_SetProperties;
    NBFIF_GET_STAT_CAP           NBFIF_GetStatCapability;
    NBFIF_GET_STAT               NBFIF_GetStatistics;
    NBFIF_CNTRL_STAT             NBFIF_ControlStatEngine;
} NBFIF_REGISTER_PARAM, *PNBFIF_REGISTER_PARAM;
```

Element	Definition
hDev	See "_NBFIF_DEV_HANDLE Type" on page 30.
peId	
ifId	
list of function pointers	Pointers to your implemented functions. If you choose not to implement any one or more of the optional functions, you must pass NULL instead of that function's pointer.

Description

Use this structure when calling IOCTL_NBOOST_FOREIGN_REGISTER.

See Also

- "IOCTL_NBOOST_FOREIGN_REGISTER Function" on page 83
- "Initiating Communication With the Policy Accelerator" on page 12

_NBFIF_STAT_CAP_ITEM Structure

Describes statistical items supported by your NIC and associates an index number with each item.

```
typedef struct _NBFIF_STAT_CAP_ITEM {
     CHAR   statName [NBFIF_MAX_NAME];
     UINT32 statIndx;
       } NBFIF_STAT_CAP_ITEM, *PNBFIF_STAT_CAP_ITEM;
```

Element	Definition
statName	A name of your choice that describes a statistical item.
statIndx	An index number for the statistical item within the table of items. The NIC driver can use any arbitrary numbering scheme for this index.

Description

Fill in this table to describe the statistical items supported by your NIC. For example, this structure looks as follows for three statistical items:

NBFIF_STAT_CAP_ITEM
Index #
NAME

NBFIF_STAT_CAP_ITEM
Index #
NAME

NBFIF_STAT_CAP_ITEM
Index #
NAME

See Also

- “Implementing the NIC’s Statistical Functions” on page 25
- “NBFIF_GetStatCapability Function” on page 58

_NBFIF_STAT_CNTRL Enumeration

Provides valid settings for turning the NIC's statistical calculations on and off.

```
typedef enum _NBFIF_STAT_CNTRL {
     NBFIF_STAT_ON,
     NBFIF_STAT_OFF
            } NBFIF_STAT_CNTRL;
```

Element	Definition
NBFIF_STAT_ON	Turns statistics on.
NBFIF_STAT_OFF	Turns statistics off.

Description

This enumeration applies only to the NBFIF_ControlStatEngine Function.

See Also

- "Implementing the NIC's Statistical Functions" on page 25
- "NBFIF_ControlStatEngine Function" on page 53

NBFIF_STATUS Type

A status code that each of your functions must return, indicating the success or failure of the function.

```
typedef UINT32 NBFIF_STATUS;

#define NBFIF_STATUS_SUCCESS                0x00000000L
#define NBFIF_INVALID_HANDLE                0xE000F001L
#define NBFIF_OUT_OF_MEMORY                 0xE000F002L
#define NBFIF_BUFFER_TYPE_NOT_SUPPORTED     0xE000F003L
#define NBFIF_HARDWARE_NOT_READY            0xE000F004L
#define NBFIF_OUT_OF_HARDWARE_RESOURCE      0xE000F005L
#define NBFIF_BUFFER_OUT_OF_RANGE           0xE000F006L
#define NBFIF_1_OR_MORE_PROPERTIES_INVALID  0xE000F007L
#define NBFIF_OUT_OF_RX_BUFFER              0xE000F008L
```

Description

Each of the functions that you implement must return a status code indicating whether the function succeeded. The header file `nbfif.h` defines a UINT32 type named `NBFIF_STATUS` to assist with this.

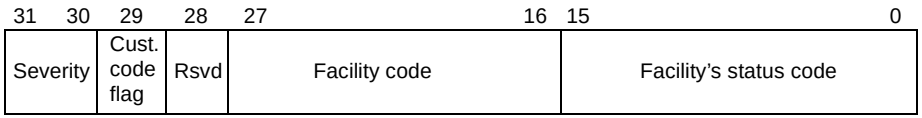
NOTE: The PA-100 driver evaluates your returned status only for success or failure; it does not take any actions based on any particular failure, but rather simply passes the status up to the calling application if applicable.

NOTE: The `NBFIF_STATUS` type conforms to the Windows NT operating system's standard 32-bit error code format. Refer to your operating system documentation for details. The information here is provided as a summary of that information.

Windows NT Error Code Format

The Windows NT error code format contains 32 bits as follows:

Bit number:



Starting Bit	Length	Content	Description
31	2	Severity	The severity of the status message being reported: ■ 0: Success ■ 1: Informational; the calling code can ignore this message ■ 2: Warning ■ 3: Error
29	1	Customer code flag	
28	1	Reserved	
27	12	Facility code	
15	16	Facility status code	

Predefined Status Values

The header file also defines a small set of useful 32-bit statuses, as shown in the syntax, that you can return. You do not have to use any of these values; you can define your own.

See Also

- "Reporting and Evaluating Status" on page 23

_NBFIF_UNREGISTER_PARAM Structure

The NIC driver's description of its ODX interface for use in unregistering itself from the PA-100 driver.

```
/*
 * This is the input parameter for calling
IOCTL_NBOOST_FOREIGN_UNREGISTER
 * as defined in nbpe.h
 */
typedef struct _NBFIF_UNREGISTER_PARAM
{
    NBFIF_DEV_HANDLE    hFifDev;
    NBPE_DEV_HANDLE     hNbpeDev;
} NBFIF_UNREGISTER_PARAM, *PNBFIF_UNREGISTER_PARAM;
```

Element	Definition
hFifDev	
hNbpeDev	

Description

Use this structure when calling IOCTL_NBOOST_FOREIGN_UNREGISTER.

See Also

- "IOCTL_NBOOST_FOREIGN_UNREGISTER Function" on page 84
- "Terminating Communication With the Policy Accelerator" on page 15

NBFIF Functions Alphabetic Reference

NBFIF Functions Summary

The `nbfif.h` header file contains prototypes in the C language for the following functions, which you must implement:

Function	Description
`NBFIF_BindRecv` Function	Required. Implement this function to bind a Policy Accelerator to receive packet traffic from the NIC and to start the NIC.
`NBFIF_BoundTxPacketsReady` Function	Optional. Implement this optional function to transmit all buffers from the PA-100 driver's buffer ring.
`NBFIF_ControlStatEngine` Function	Optional. Implement this optional function to turn the NIC's hardware statistics calculations on or off.
`NBFIF_GetPropCapability` Function	Optional. Implement this optional function to list which properties your NIC driver supports.
`NBFIF_GetProperties` Function	Optional Implement this optional function to provide the current settings of the requested NIC properties to the PA-100 driver.
`NBFIF_GetStatCapability` Function	Optional. Implement this optional function to list which statistical items your NIC driver supports.
`NBFIF_GetStatistics` Function	Optional. Implement this optional function to provide the requested statistical information about the NIC to the PA-100 driver.
`NBFIF_PacketReady` Function	Required. Implement this required function to allow the PA-100 driver to tell the NIC driver that there is a packet ready for transfer from the Policy Accelerator to the NIC and provide a packet buffer.
`NBFIF_Reset` Function	Optional but recommended. Implement this optional function to reset the NIC driver and NIC.

Function	Description
`NBFIF_SetProperties` Function	Optional. Implement this optional function to set values for a specified set of NIC properties.
`NBFIF_UnbindRecv` Function	Required. Implement this required function to stop the NIC from sending packets to the Policy Accelerator.

NBFIF_BindRecv Function

Implement this function to bind a Policy Accelerator to receive packet traffic from the NIC and to start the NIC.

```
NBFIF_STATUS NBFIF_BindRecv (
          NBFIF_DEV_HANDLE      hDev,
          PNBPE_BIND_RECV_PARAM pNBPE );
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver. See "_NBFIF_DEV_HANDLE Type" on page 30.
pNBPE	A set of pointers to the functions in the PA-100 driver. The PA-100 driver returns these pointers based on the specific Policy Accelerator to which the NIC driver has initiated communication. See "_NBPE_BIND_RECV_PARAM Structure" on page 72.

Returns

If successful, return the predefined status value NBFIF_STATUS_SUCCESS. If the NIC driver cannot complete the operation, you can return any error code.

Description

The PA-100 driver calls this function after the NIC driver registers itself using IOCTL_NBOOST_FOREIGN_REGISTER.

The primary purpose of this function is to get the PA-100 driver function pointers to use later. Therefore, in you NIC driver, you must implement this function so that it does the following:

1. Save for future use the function pointers passed to it by the PA-100 driver in the pNBPE argument.
2. Return a status based on the success of this function.

In addition, this can be a logical place to do such things as to redirect traffic from the NIC's receiving buffers to the Policy Accelerator's buffers and to restart the NIC. So, for example, you could also do the following in this function:

1. Verify that the NIC hardware is halted and, if not, halt it.
2. Redirect traffic from the NIC's buffers to the Policy Accelerator's buffers as follows:
   a. Call NBPE_GetBufferTypes to get a list of valid buffer types.
   b. Select the buffer type that is appropriate for your NIC.

c. Call `NBPE_GetRecvPEBufRing` to get the location of the Policy Accelerator's empty packet buffers.

d. Do any initialization required to start the packet flow.

3. Start the NIC hardware.

   If all initialization has been successful, packets begin flowing.

NOTE: You do not need to set up buffers in `NBFIF_BindRecv`, but you must do it at some point to enable the NIC to receive buffers on behalf of the Policy Accelerator.

See Also

- "Initiating Communication With the Policy Accelerator" on page 12
- "NBFIF_UnbindRecv Function" on page 65
- "NBPE_GetRecvPEBufRing Function" on page 86
- "_NBPE_BIND_RECV_PARAM Structure" on page 72
- "NBPE_STATUS Type" on page 79

NBFIF_BoundTxPacketsReady Function

Implement this optional function to transmit all buffers from the PA-100 driver's buffer ring.

```
NBFIF_STATUS NBFIF_BoundTxPacketsReady (
        NBFIF_DEV_HANDLE hDev);
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_-REGISTER. See "_NBFIF_DEV_HANDLE Type" on page 30.

Returns

If successful, return the predefined status value NBFIF_STATUS_SUCCESS. If the NIC driver cannot complete the operation, you can return any error code.

Description

This function transmits packets from the Policy Accelerator through the NIC using the PA-100 driver's transmit ring buffer rather than transmitting packets from the Policy Accelerator one at a time.

NOTE: If you choose not to implement this function, you must pass a NULL in place of a pointer to this function when calling IOCTL_NBOOST_-FOREIGN_REGISTER.

NOTE: This is one of two functions that allow the Policy Accelerator to transmit packets through the NIC. This is the faster function, but it is optional. If your NIC driver does not implement this function, the PA-100 driver uses NBFIF_PacketReady.

The PA-100 driver calls this function to notify the NIC driver that the Policy Accelerator's transmit buffer ring contains packets that are ready to be transmitted.

In your NIC driver, you must implement this function so that it does the following:

1. Transmits packets from the transmit buffer ring.
2. Return a status based on the success of these operations.

See Also

- "Initiating Communication With the Policy Accelerator" on page 12
- "Handling Packets" on page 17

- “NBFIF_PacketReady Function” on page 62
- “NBPE_STATUS Type” on page 79

NBFIF_ControlStatEngine Function

Implement this optional function to turn the NIC's hardware statistics calculations on or off.

```
NBFIF_STATUS NBFIF_ControlStatEngine (
          NBFIF_DEV_HANDLE hDev,
          NBFIF_STAT_CNTRL cntrl );
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_REGISTER. See "_NBFIF_DEV_HANDLE Type" on page 30.
cntrl	The PA-100 driver uses this to specify whether to turn the statistics on or off, as described in "_NBFIF_STAT_CNTRL Enumeration" on page 43.

Returns

If successful, return the predefined status value NBFIF_STATUS_SUCCESS. If the NIC driver cannot complete the operation, you can return any error code.

Description

The PA-100 driver calls this function when an application requests that the NIC collect or stop collecting statistical information. Statistical information might include such items as the quantity of transmit or receive errors; the availability of statistical items depends entirely on the features of your NIC and your implementation of the statistics functions in this library.

NOTE: If you choose not to implement this function, you must pass a NULL in place of a pointer to this function when calling IOCTL_NBOOST_FOREIGN_REGISTER.
In this case, the PA-100 drivers assume that statistical calculations are always off.

If your NIC driver is capable of providing statistical information, implement this function so that it does the following:

1. If your NIC driver allows statistical calculations to be turned on and off, turn the NIC's hardware statistics engine on or off based on the requested setting of *cntrl*. If statistics cannot be turned off, simply return a success indicator in the following step.
2. Return a status based on the success of these operations.

See Also

- "Implementing the NIC's Statistical Functions" on page 25
- "NBPE_STATUS Type" on page 79

NBFIF_GetPropCapability Function

Implement this optional function to list which properties your NIC driver supports.

```
NBFIF_STATUS NBFIF_GetPropCapability (
         NBFIF_DEV_HANDLE     hDev,
         UINT32              *pNumProp,
         PNBFIF_PROP_CAP_ITEM pPropCapList );
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_-REGISTER. See "_NBFIF_DEV_HANDLE Type" on page 30.
pNumProp	A count, provided by your NIC driver, of the quantity of supported properties.
pPropCapList	A table in which your NIC driver provides information about all supported NIC properties. See "_NBFIF_PROP_CAP_ITEM Structure" on page 34. If the PA-100 driver does not need this information, it passes a NULL.

Returns

If successful, return the predefined status value NBFIF_STATUS_SUCCESS. If the NIC driver cannot complete the operation, you can return any error code.

Description

This function provides a table that lists, for each property supported in the NIC driver, the property's name, a corresponding property index, its possible values, and whether the property can be viewed or changed.

NOTE: If you choose not to implement this function, you must pass a NULL in place of a pointer to this function when calling IOCTL_NBOOST_-FOREIGN_REGISTER.

The PA-100 driver calls this function during initialization to determine what properties are available in the NIC for viewing and setting by an IXA application. The PA-100 driver itself does not use any of the property information.

In your NIC driver, you must implement this function so that it does the following:

1. Provide information about supported properties as follows:
   - Return in *pNumProp* the count of the properties that the NIC driver supports.
   - If *pPropCapList* is NULL, do nothing else, otherwise, fill in the *pPropCapList* table as described in "_NBFIF_PROP_CAP_ITEM Structure" on page 34.

     You provide a table that lists, for each property of the NIC driver, the property's name and a corresponding index number.
2. Return a status based on the success of this operation.

See Also

- "Implementing the NIC's Properties Functions" on page 23
- "NBPE_STATUS Type" on page 79

NBFIF_GetProperties Function

Implement this optional function to provide the current settings of the requested NIC properties to the PA-100 driver.

```
NBFIF_STATUS NBFIF_GetProperties (
          NBFIF_DEV_HANDLE          hDev,
          UINT32                    numProp,
          PNBFIF_GET_SET_PROP_ITEM  pGetPropList );
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_-REGISTER. See "_NBFIF_DEV_HANDLE Type" on page 30.
numProp	Provided by the PA-100 driver. The quantity of properties that the PA-100 driver has included in *pGetPropList*.
pGetPropList	A pointer to a table that contains one or more property indexes, as defined by the "NBFIF_GetPropCapability Function" on page 54, about which the PA-100 driver is requesting information, and space in which the NIC driver should return the properties' current settings. See "_NBFIF_GET_SET_PROP_ITEM Structure" on page 31.

Returns

If successful, return the predefined status value NBFIF_STATUS_SUCCESS. If the NIC driver cannot complete the operation, you can return any error code.

Description

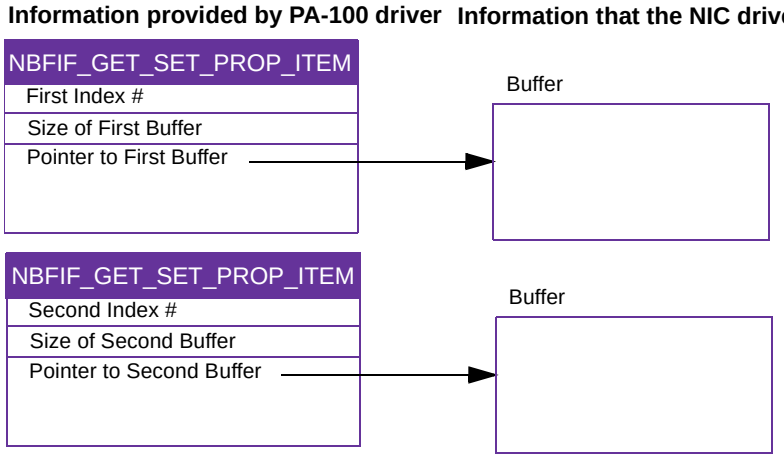
The PA-100 driver calls this function when an IXA application requests information about the current settings of one or more of the NIC's properties.

NOTE: If you choose not to implement this function, you must pass a NULL in place of a pointer to this function when calling IOCTL_NBOOST_-FOREIGN_REGISTER.

In your NIC driver, you must implement this function so that it does the following:

1. Provide the current settings for the requested properties.

   For example, the following diagram shows a table containing two property indexes and space in which you provide the properties' values:



2. Return a status based on the success of these operations.

   For example, if any of the index numbers are invalid, you could return the predefined status value `NBFIF_1_OR_MORE_PROPERTIES_INVALID`.

See Also

- “Implementing the NIC’s Properties Functions” on page 23
- “_NBFIF_GET_SET_PROP_ITEM Structure” on page 31
- “NBPE_STATUS Type” on page 79

NBFIF_GetStatCapability Function

Implement this optional function to list which statistical items your NIC driver supports.

```
NBFIF_STATUS NBFIF_GetStatCapability (
          NBFIF_DEV_HANDLE     hDev,
          UINT32              *pNumStat,
          PNBFIF_STAT_CAP_ITEM pStatCapList );
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_-REGISTER. See "_NBFIF_DEV_HANDLE Type" on page 30.
pNumStat	A count, provided by your NIC driver, of the quantity of supported statistical items.
pStatCapList	A table in which your NIC driver provides information about all supported NIC statistical items. See "_NBFIF_STAT_CAP_ITEM Structure" on page 42. If the PA-100 driver does not need this information, it passes a NULL.

Returns

If successful, return the predefined status value NBFIF_STATUS_SUCCESS. If the NIC driver cannot complete the operation, you can return any error code.

Description

The PA-100 driver calls this function when an IXA application requests what statistical items are available in the NIC for viewing by the application. The PA-100 driver itself does not use any of the statistical information.

NOTE: If you choose not to implement this function, you must pass a NULL in place of a pointer to this function when calling IOCTL_NBOOST_-FOREIGN_REGISTER.

In your NIC driver, you must implement this function so that it does the following:

1. Provide information about supported statistical items as follows:
   - Return in *pNumStat* the count of the statistical items that the NIC driver supports.
   - If *pStatCapList* is NULL, do nothing else, otherwise, fill in the *pStat-CapList* table as described in "_NBFIF_STAT_CAP_ITEM Structure" on page 42.

You provide a table that lists, for each statistical measurement supported in the NIC driver, the statistic's name and a corresponding index number.

2. Return a status based on the success of these operations.

See Also

- "Implementing the NIC's Statistical Functions" on page 25
- "_NBFIF_STAT_CAP_ITEM Structure" on page 42
- "NBFIF_ControlStatEngine Function" on page 53
- "NBPE_STATUS Type" on page 79

NBFIF_GetStatistics Function

Implement this optional function to provide the requested statistical information about the NIC to the PA-100 driver.

```
NBFIF_STATUS NBFIF_GetStatistics (
          NBFIF_DEV_HANDLE     hDev,
          UINT32               numStat,
          PNBFIF_GET_STAT_ITEM pGetStatList );
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_-REGISTER. See "_NBFIF_DEV_HANDLE Type" on page 30.
numStat	Provided by the PA-100 driver. The quantity of statistical items that the PA-100 driver has included in pGetStatList.
pGetStatList	A table containing one or more statistical indexes, as set by the "NBFIF_GetStatCapability Function" on page 58, and space in which the NIC driver should return the statistics' current values. See "_NBFIF_GET_STAT_ITEM Structure" on page 33.

Returns

If successful, return the predefined status value NBFIF_STATUS_SUCCESS. If the NIC driver cannot complete the operation, you can return any error code.

Description

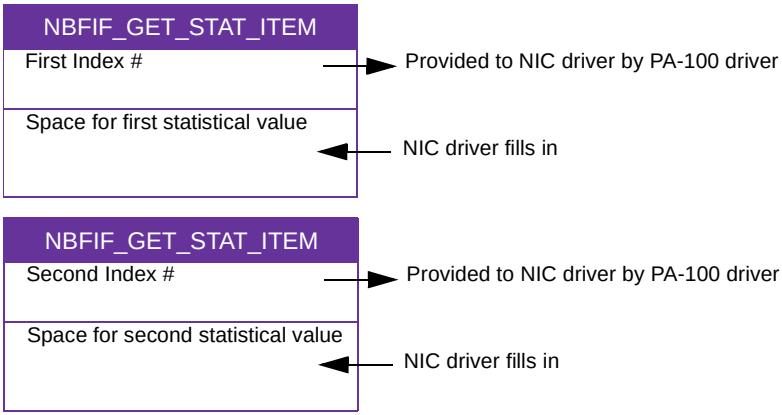
The PA-100 driver calls this function when an IXA application requests statistical information about the NIC.

NOTE: If you choose not to implement this function, you must pass a NULL in place of a pointer to this function when calling IOCTL_NBOOST_-FOREIGN_REGISTER.

In your NIC driver, you must implement this function so that it does the following:

1. Provide the current settings for the requested statistics.

   For example, the following diagram shows a table containing two statistic indexes and space in which you provide the statistics' values:



2. Return a status based on the success of these operations.

This function fills in the counter value of each interested statistic items in the input array. Upon receiving this call, the NIC driver reads the current statistic count out of the NIC card and fills in the appropriate array entry.

See Also

- “Implementing the NIC’s Statistical Functions” on page 25
- “_NBFIF_GET_STAT_ITEM Structure” on page 33
- “NBPE_STATUS Type” on page 79

NBFIF_PacketReady Function

Implement this required function to allow the PA-100 driver to tell the NIC driver that there is a packet ready for transfer from the Policy Accelerator to the NIC and provide a packet buffer.

```
NBFIF_STATUS NBFIF_PacketReady (
          NBFIF_DEV_HANDLE hDev,
          PHY_ADDR        *pPacket,
          UINT32           sizeInBytes );
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_-REGISTER. See "_NBFIF_DEV_HANDLE Type" on page 30.
pPacket	The PA-100 driver passes this pointer to the packet that is ready for transmission.
sizeInBytes	The PA-100 driver passes the size of the packet.

Returns

If successful, return the predefined status value NBFIF_STATUS_SUCCESS. If the NIC driver cannot complete the operation, you can return any error code.

Description

The PA-100 driver calls this function to transmit a packet out through the NIC.

NOTE: This is one of two functions that allow the Policy Accelerator to transmit packets out through the NIC. This is the default, slower function. If your NIC driver implements the NBFIF_BoundTxPacketsReady function, the PA-100 driver uses that function **instead** of NBFIF_PacketReady.

In your NIC driver, you must implement this function so that it does the following:

1. Transmit the packet.
2. Return a status based on the success of these operations.

See Also

- "Handling Packets" on page 17
- "NBPE_STATUS Type" on page 79

NBFIF_Reset Function

Implement this optional function to reset the NIC driver and NIC.

```
VOID NBFIF_Reset ( NBFIF_DEV_HANDLE hDev );
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_REGISTER. See "_NBFIF_DEV_HANDLE Type" on page 30.

Returns

If successful, return the predefined status value NBFIF_STATUS_SUCCESS. If the NIC driver cannot complete the operation, you can return any error code.

Description

The PA-100 driver calls this function when it needs to reset the NIC and NIC driver to a known, clean state.

This function is optional, but implementation is recommended.

NOTE: If you choose not to implement this function, you must pass a NULL in place of a pointer to this function when calling IOCTL_NBOOST_-FOREIGN_REGISTER.

In your NIC driver, you must implement this function so that it does the following:

1. Reset the NIC driver.
2. Reset the NIC
3. Abort all ongoing transmission and receiving of packets.
4. Return a status based on the success of these operations.

See Also

- "Initiating Communication With the Policy Accelerator" on page 12
- "NBPE_STATUS Type" on page 79

NBFIF_SetProperties Function

Implement this optional function to set values for a specified set of NIC properties.

```
NBFIF_STATUS NBFIF_SetProperties (
          NBFIF_DEV_HANDLE          hDev,
          UINT32                    numProp,
          PNBFIF_GET_SET_PROP_ITEM  pSetPropList );
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_-REGISTER. See "_NBFIF_DEV_HANDLE Type" on page 30.
numProp	Provided by the PA-100 driver. The quantity of properties that the PA-100 driver has included in pSetPropList.
pSetPropList	A table containing one or more property indexes, as set by the "NBFIF_GetPropCapability Function" on page 54, and the values that the NIC driver should use to change the properties' settings. See "_NBFIF_GET_SET_PROP_ITEM Structure" on page 31.

Returns

If successful, return the predefined status value NBFIF_STATUS_SUCCESS. If the NIC driver cannot complete the operation, you can return any error code.

Description

The PA-100 driver calls this function when an IXA application requests a change in the settings of one or more of the NIC's properties.

NOTE: If you choose not to implement this function, you must pass a NULL in place of a pointer to this function when calling IOCTL_NBOOST_-FOREIGN_REGISTER.

In your NIC driver, you must implement this function so that it does the following:

1. Change the settings of the indicated properties.
2. Return a status based on the success of these operations.

   For example, if any of the index numbers are invalid, you could return the predefined status value NBFIF_1_OR_MORE_PROPERTIES_INVALID.

See Also

- "Implementing the NIC's Properties Functions" on page 23
- "NBPE_STATUS Type" on page 79

NBFIF_UnbindRecv Function

Implement this required function to stop the NIC from sending packets to the Policy Accelerator.

```
NBFIF_STATUS NBFIF_UnbindRecv ( NBFIF_DEV_HANDLE hDev );
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_REGISTER. See "_NBFIF_DEV_HANDLE Type" on page 30.

Returns

If successful, return the predefined status value NBFIF_STATUS_SUCCESS. If the NIC driver cannot complete the operation, you can return any error code.

Description

The PA-100 driver calls this function to stop receiving packet traffic from the NIC.

This function must cleanly terminate its connection to the Policy Accelerator. Recommended steps include:

1. Stop sending packets to the Policy Accelerator.
2. Free all buffers in the NIC, if any exist.
3. Return all allocated buffers to the Policy Accelerator using NBPE_ReturnXmitPEBuffers.
4. NULL the PA-100 driver function pointers that the NIC driver's NBFIF_BindRecv function saved.

   This ensures that the PA-100 driver does not accidentally attempt to use functions that are now invalid.
5. Stop the NIC hardware
6. Return a status based on the success of these operations.

NOTE: After the PA-100 driver calls this function, the NIC driver can no longer call any of the PA-100 driver functions.

See Also

- "Terminating Communication With the Policy Accelerator" on page 15
- "NBPE_STATUS Type" on page 79

Chapter 4
Alphabetic Reference: NBPE (PA-100 Driver) Functions, Types, and Structures

This chapter contains an alphabetic reference to the Intel® Optimal Data Exchange (ODX) Protocol for PCI functions that your NIC driver uses to communicate with the PA-100 driver, and to the types and structures that you use within those functions.

It describes each function and data element, and explains in detail how to use each function as part of your NIC driver customization.

It contains the following sections:

Overview

The `nbpe.h` header file provides the prototypes of Policy Accelerator driver functions that your NIC driver will use. Your driver code must include this file.

CAUTION: Do not modify this header file.

NBPE Types and Structures Alphabetic Reference

NBPE Types and Structures Summary

The nbpe.h header file contains the following data types and structures that the NIC driver and the PA-100 driver use to pass information between themselves:

Data Element	Description
_NBPE_ANIC_RX_RING Structure	Provides the location of the packet buffer that is available for the NIC to place packets into that it receives on behalf of the Policy Accelerator.
_NBPE_ANIC_TX_RING Structure	Provides the location and size of the packet ring buffer that contains packets for the NIC to transmit on behalf of the Policy Accelerator.
_NBPE_BIND_RECV_PARAM Structure	Description of the PA-100 driver's ODX interface for use in establishing the connection between the NIC driver and the PA-100 driver.
_NBPE_BUF_TYPE_DETAIL Structure	Describes the types of packet buffers that the Policy Accelerator supports.
NBPE_DEV_HANDLE Type	A unique handle.
NBPE_DEVICE_NAME Define	Hard-coded device name for the PA-100 driver.
NBPE_LINK_NAME Define	Link name for the PA-100 driver.
_NBPE_PKT_READY_INFO Structure	Describes the packets that the NIC has received on behalf of the Policy Accelerator and that are ready for the Policy Accelerator.
_NBPE_RX_RING_INFO Structure	Describes the packet buffer ring that is available for the NIC to place packets that it receives on behalf of the Policy Accelerator.

Data Element	Description
NBPE_STATUS Type	A status code that each of the PA-100 functions returns, indicating the success or failure of the function.
_NBPE_TX_RING_INFO Structure	Describes the packet buffer ring that contains packets for the NIC to transmit on behalf of the Policy Accelerator.

_NBPE_ANIC_RX_RING Structure

Provides the location of the packet buffer that is available for the NIC to place packets into that it receives on behalf of the Policy Accelerator.

```
typedef struct _NBPE_ANIC_RX_RING {
   ULONG      bufAddr;
} NBPE_ANIC_RX_RING, *PNBPE_ANIC_RX_RING;
```

Element	Definition
bufAddr	The address on the Policy Accelerator of ring buffer space for packets that the NIC receives. This address points to the Ethernet header of the packet and is in appropriate format for the platform on which the drivers are running.

See Also

- "_NBPE_RX_RING_INFO Structure" on page 78
- "NBPE_GetRecvPEBufRing Function" on page 86

_NBPE_ANIC_TX_RING Structure

Provides the location and size of the packet ring buffer that contains packets for the NIC to transmit on behalf of the Policy Accelerator.

```
typedef struct _NBPE_ANIC_TX_RING{
   ULONG      bufAddr;
   ULONG      byteLen;
} NBPE_ANIC_TX_RING, *PNBPE_ANIC_TX_RING;
```

Element	Definition
bufAddr	The address on the Policy Accelerator of ring buffer space for a packet that is available for the NIC to transmit. This address points to the Ethernet header of the first available packet and is in appropriate format for the platform on which the drivers are running.
byteLen	The quantity of bytes in a packet. Not including CRC

See Also

- "_NBPE_TX_RING_INFO Structure" on page 81
- "NBPE_GetTransmitPEBufRing Function" on page 87

_NBPE_BIND_RECV_PARAM Structure

Description of the PA-100 driver's ODX interface for use in establishing the connection between the NIC driver and the PA-100 driver.

```
typedef struct _NBPE_BIND_RECV_PARAM {
    NBPE_DEV_HANDLE         hDev;
         /* Actual function pointers */
    NBPE_RESET              NBPE_Reset;
    NBPE_GET_BUF_TYPES      NBPE_GetBufferTypes;
    NBPE_GET_RX_PE_BUF_RING NBPE_GetRecvPEBufRing;
    NBPE_GET_TX_PE_BUF_RING NBPE_GetTransmitPEBufRing;
    NBPE_RETURN_XMIT_BUF    NBPE_ReturnXmitPEBuffer;
    NBPE_PACKET_READY       NBPE_PacketReady;
} NBPE_BIND_RECV_PARAM, *PNBPE_BIND_RECV_PARAM;
```

Element	Definition
hDev	See "NBPE_DEV_HANDLE Type" on page 74.
list of function pointers	Pointers to the PA-100 driver functions. Your driver must save these pointers.

See Also

- "NBFIF_BindRecv Function" on page 49
- "NBPE_GetBufferTypes Function" on page 85

_NBPE_BUF_TYPE_DETAIL Structure

Describes the types of packet buffers that the Policy Accelerator supports.

```
typedef struct _NBPE_BUF_TYPE_DETAIL{
   UINT32      bufType;
   PHY_ADDR    validAddrMask;
   UINT32      bufAlign;
   UINT32      packetStartOffset;
   UINT32      maxPacketSize; /
} NBPE_BUF_TYPE_DETAIL, *PNBPE_BUF_TYPE_DETAIL;
```

Element	Definition
bufType	A unique identifier for this buffer type. The NIC driver uses this in the NBPE_GetRecvPEBufRing Function and NBPE_GetTransmitPEBufRing Function.
validAddrMask	Mask to the address that determines whether the buffer is within range.
bufAlign	The alignment of the buffer in number of bytes; for example, 2048 for a 2-KB-aligned buffer. /
packetStartOffset	Byte offset from beginning of the aligned buffer that the Ethernet packet must start from.
maxPacketSize	The maximum payload allowed in this buffer type, in bytes.

Description

The buffer location before and after the offset and the maximum payload are reserved values; the NIC driver must not use them.

See Also

- “NBPE_GetBufferTypes Function” on page 85

NBPE_DEV_HANDLE Type

A unique handle.

Description

The PA-100 driver uses the pointer only to identify the NIC driver from which it is receiving a request. Therefore, you must use the same pointer in successive calls to other PA-100 driver functions.

NBPE_DEVICE_NAME Define

Hard-coded device name for the PA-100 driver.

```
#define NBPE_DEVICE_NAME   L"\\Device\\nboost"
```

NBPE_LINK_NAME Define

Link name for the PA-100 driver.

```
#define NBPE_LINK_NAME L"\\DosDevices\\nboost"
```

_NBPE_PKT_READY_INFO Structure

Describes the packets that the NIC has received on behalf of the Policy Accelerator and that are ready for the Policy Accelerator.

```
typedef struct _NBPE_PKT_READY_INFO {
   union _rxInfo
      { struct _layout
         { UINT16    pktLen;
           UINT16    tcpCheckSum;
         } layout;

         UINT32      rxInfoWord;
      } rxInfo;
} NBPE_PKT_READY_INFO, *PNBPE_PKT_READY_INFO;
```

Element	Definition
pktLen	
tcpCheckSum	
rxInfoWord	

See Also

- "NBPE_PacketReady Function" on page 88

_NBPE_RX_RING_INFO Structure

Describes the packet buffer ring that is available for the NIC to place packets that it receives on behalf of the Policy Accelerator.

```
typedef struct _NBPE_RX_RING_INFO {
   PNBPE_ANIC_RX_RING   pRingBase;
   UINT32               indxMask;
   UINT32              *pProdIndx;
}NBPE_RX_RING_INFO, *PNBPE_RX_RING_INFO;
```

Element	Definition
pRingBase	The location on the Policy Accelerator of the buffer ring.
indxMask	Apply this mask to the produce index to ensure that wraps through the ring buffer are accounted for.
pProdIndx	The *produce index;* that is, the address on the Policy Accelerator of where the NIC can place packets. This address is in appropriate format for the platform on which the drivers are running.

See Also

- "_NBPE_ANIC_RX_RING Structure" on page 70
- "NBPE_GetRecvPEBufRing Function" on page 86

NBPE_STATUS Type

A status code that each of the PA-100 functions returns, indicating the success or failure of the function.

```
typedef UINT32 NBFIF_STATUS;

#define NBPE_STATUS_SUCCESS                  0x00000000L
#define NBPE_INVALID_HANDLE                  0xE000E001L
#define NBPE_NOT_ENOUGH_BUFFERS              0xE000E002L
#define NBPE_RETURN_BUFFER_OUT_OF_SEQUENCE   0xE000E003L
#define NBPE_BUF_TYPE_NOT_SUPPORTED          0xE000e004L
```

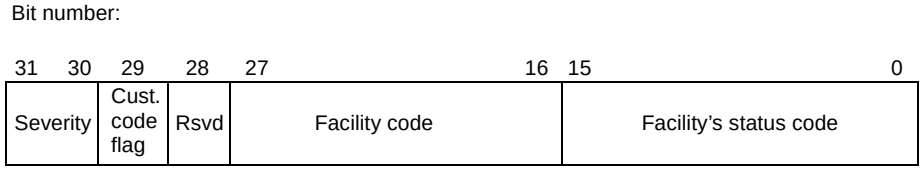
Description

Each of the PA-100 functions that your NIC driver calls returns a status code indicating whether the function succeeded. The header file `nbpe.h` defines a UINT32 type named `NBPE_STATUS`.

NOTE: The `NBPE_STATUS` type conforms to the Windows NT operating system's standard 32-bit error code format. Refer to your operating system documentation for details. The information here is provided as a summary of that information.

Windows NT Error Code Format

The Windows NT error code format contains 32 bits as follows:



Starting Bit	Length	Content	Description
31	2	Severity	The severity of the status message being reported: ■ 0: Success ■ 1: Informational; the calling code can ignore this message ■ 2: Warning ■ 3: Error
29	1	Customer code flag	
28	1	Reserved	
27	12	Facility code	
15	16	Facility status code	

Predefined Status Values

The header file also defines a small set of useful 32-bit statuses, as shown in the syntax, that might be returned to your NIC driver.

See Also

- "Reporting and Evaluating Status" on page 23

_NBPE_TX_RING_INFO Structure

Describes the packet buffer ring that contains packets for the NIC to transmit on behalf of the Policy Accelerator.

```
typedef struct _NBPE_TX_RING_INFO {
   PNBPE_ANIC_TX_RING   pRingBase;
   UINT32               indxMask;
   UINT32              *pProdIndx;
}NBPE_TX_RING_INFO, *PNBPE_TX_RING_INFO;
```

Element	Definition
pRingBase	The location on the Policy Accelerator of the buffer ring.
indxMask	Apply this mask to the produce index to ensure that wraps through the ring buffer are accounted for.
pProdIndx	The *produce index;* that is, the address on the Policy Accelerator of the end point of where in the buffer ring the Policy Accelerator has placed packets. This address is in appropriate format for the platform on which the drivers are running.

See Also

- "_NBPE_ANIC_TX_RING Structure" on page 71
- "NBPE_GetTransmitPEBufRing Function" on page 87

NBPE Functions Alphabetic Reference

NBPE Functions Summary

The nbhwpe.h header file contains prototypes for the following functions, which your driver might/must call:

Function	Description
IOCTL_NBOOST_FOREIGN_REGISTER Function	Use this function to open communication between your NIC driver and the PA-100 drivers.
IOCTL_NBOOST_FOREIGN_UNREGISTER Function	Use this function to disconnect the NIC driver from the PA-100 driver.
NBPE_GetBufferTypes Function	Use this function to receive a list of the types of packet buffers that the Policy Accelerator supports.
NBPE_GetRecvPEBufRing Function	Use this function to determine the location of a buffer ring in which to put packets that the NIC receives on behalf of the Policy Accelerator.
NBPE_GetTransmitPEBufRing Function	Use this function to determine the location of a buffer ring for packets that the NIC will transmit on behalf of the Policy Accelerator.
NBPE_PacketReady Function	Use this function to notify the PA-100 driver that received packets are ready in the buffer ring.
NBPE_Reset Function	Use this function to reset the Policy Accelerator.
NBPE_ReturnXmitPEBuffer Function	Use this function to return transmitted buffers to the Policy Accelerator.

IOCTL_NBOOST_FOREIGN_REGISTER Function

Use this function to open communication between your NIC driver and the PA-100 drivers.

```
#define FILE_DEVICE_NBOOST        0x000083001
#define NBOOST_FUNC_FOREIGN_REG   0x0964
#define NBOOST_FUNC_FOREIGN_UNREG 0x0965

IOCTL_NBOOST_FOREIGN_REGISTER
          CTL_CODE (FILE_DEVICE_NBOOST,
                    NBOOST_FUNC_FOREIGN_REG,
                    METHOD_BUFFERED,
                    FILE_ANY_ACCESS)
```

Argument	Description
FILE_DEVICE_NBOOST NBOOST_FUNC_FOREIGN_REG	These functions are strictly used by the NIC connection only. These number SHOULD NOT be changed because the same definition is mirrored in the internal .h file. FILE_DEVICE_NBOOST is the PA-100 driver.

Description

Your NIC driver must call this function before it can call any other PA-100 (NBPE) functions. It opens the Policy Accelerator identified by the device defines.

This is the first command that NIC driver must call. It opens the chosen PA-100 device. The NIC driver calls this IOCTL command with a set of function pointers described in “_NBFIF_REGISTER_PARAM Structure” on page 41.

Upon receiving this IOCTL command, the PA-100 driver registers the NIC driver into the system. In return, the PA-100 driver returns status success or failure to the caller.

Everything is standard Windows NT IOCTL stuff otherwise.

See Also

- “Initiating Communication With the Policy Accelerator” on page 12

IOCTL_NBOOST_FOREIGN_UNREGISTER Function

Use this function to disconnect the NIC driver from the PA-100 driver.

```
#define FILE_DEVICE_NBOOST        0x000083001
#define NBOOST_FUNC_FOREIGN_UNREG 0x0965

#define IOCTL_NBOOST_FOREIGN_UNREGISTER
     CTL_CODE(FILE_DEVICE_NBOOST, \
              NBOOST_FUNC_FOREIGN_UNREG, \
              METHOD_BUFFERED, \
              FILE_ANY_ACCESS)
```

Description

The NIC driver must call this function to disconnect itself from the PA-100 driver. In return, the PA-100 driver calls `NBFIF_UnbindRecv`.

All PA-100 driver exported functions become inaccessible and the function pointers invalid. Moreover, the previously assigned handle is deleted.

See Also

- “Terminating Communication With the Policy Accelerator” on page 15
- “NBFIF_UnbindRecv Function” on page 65

NBPE_GetBufferTypes Function

Use this function to receive a list of the types of packet buffers that the Policy Accelerator supports.

```
NBPE_STATUS NBPE_GetBufferTypes (
          IN  NBPE_DEV_HANDLE       hDev,
          OUT UINT32               *pNumTypes,
          OUT PNBPE_BUF_TYPE_DETAIL pBufTypeList );
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_-REGISTER. See “NBPE_DEV_HANDLE Type” on page 74.
pNumTypes	A count, provided by the PA-100 driver, of the quantity of supported buffer types.
pBufTypeList	A table in which the PA-100 driver provides information about all supported buffer types. See “_NBPE_BUF_TYPE_DETAIL Structure” on page 73. If you do not need this information, pass a NULL.

Description

This function returns a table that lists, for each buffer type supported by the Policy Accelerator, such information as the buffer’s alignment, the offset into the buffer at which the Ethernet packet must start, and the maximum payload allowed in the buffer.

Your NIC driver must call this function during initialization to determine a suitable buffer type to use for passing packets between itself and the Policy Accelerator. Pass that buffer type to the NBPE_GetRecvPEBufRing Function and to the NBPE_GetTransmitPEBufRing Function.

See Also

- “Handling Packets” on page 17
- “_NBPE_BUF_TYPE_DETAIL Structure” on page 73

NBPE_GetRecvPEBufRing Function

Use this function to determine the location of a buffer ring in which to put packets that the NIC receives on behalf of the Policy Accelerator.

```
NBPE_STATUS NBPE_GetRecvPEBufRing (
          IN  NBPE_DEV_HANDLE     hDev,
          IN  UINT32              bufType,
          OUT PNBPE_RX_RING_INFO pRxRingInfo );
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_-REGISTER. See “NBPE_DEV_HANDLE Type” on page 74.
bufType	The NIC driver specifies the type of packet buffer that it wants to use; it determines this value using the NBPE_GetBufferTypes Function.
pRxRingInfo	The PA-100 driver returns a description of the assigned buffer ring in this structure. See “_NBPE_RX_RING_INFO Structure” on page 78.

Description

Your NIC driver must call this function at initialization to get the receive buffer ring. The PA-100 driver uses this ring to provide the NIC driver with empty receive buffers.

The NIC driver must maintain its own consume index to this ring, while this function provides the produce index. After the NIC driver consumes a buffer, it must increment the index by one. No additional adjustment is needed for the wraparound case. However, to get the buffer pointer, the index *must* always be masked with the index mask provided before using it. The Produce index minus the consumed index gives the number of entries actually available.

See Also

- “Initiating Communication With the Policy Accelerator” on page 12
- “_NBPE_RX_RING_INFO Structure” on page 78
- “NBPE_GetRecvPEBufRing Function” on page 86

NBPE_GetTransmitPEBufRing Function

Use this function to determine the location of a buffer ring for packets that the NIC will transmit on behalf of the Policy Accelerator.

```
NBPE_STATUS NBPE_GetTransmitPEBufRing (
          IN  NBPE_DEV_HANDLE     hDev,
          IN  UINT32              bufType,
          OUT PNBPE_TX_RING_INFO pTxRingInfo );
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_-REGISTER. See “NBPE_DEV_HANDLE Type” on page 74.
bufType	The NIC driver specifies the type of packet buffer that it wants to use; it determines this value using the NBPE_GetBufferTypes Function.
pTxRingInfo	See “_NBPE_TX_RING_INFO Structure” on page 81.

Description

The NIC driver must call this function at initialization to get the transmit buffer ring. The Policy Accelerator uses this ring to provide the bounded instance of the PA-100 driver’s transmit ring. The NIC driver can choose whether to provide a direct transmit mechanism using this ring. See NBFIF_BoundTxPacketsReady for more detail.

To access this ring, the NIC driver must maintain its own consume index to this ring, while this function provides the produce index. After the NIC driver consumes a buffer, it must increment the index by one. No additional adjustment is need for the wraparound case. However, to get the buffer pointer, the NIC driver must always mask the index with the index mask provided before using it. The Produce index minus the consumed index gives the number of entries actually available.

See Also

- “Initiating Communication With the Policy Accelerator” on page 12
- “_NBPE_TX_RING_INFO Structure” on page 81
- “NBPE_GetTransmitPEBufRing Function” on page 87

NBPE_PacketReady Function

Use this function to notify the PA-100 driver that received packets are ready in the buffer ring.

```
NBPE_STATUS NBPE_PacketReady (
          IN NBPE_DEV_HANDLE      hDev,
          IN PNBPE_PKT_READY_INFO pPacketInfo,
          IN PHY_ADDR            *pPacket);
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_-REGISTER. See “NBPE_DEV_HANDLE Type” on page 74.
pPacketInfo	See “_NBPE_PKT_READY_INFO Structure” on page 77.
pPacket	

Description

The NIC driver must use this function to tell the Policy Accelerator that a received buffer is ready. The Policy Accelerator address of the data buffer and other information that the Policy Accelerator needs to construct the media independent base header must be passed in as input arguments.

See Also

- “Handling Packets” on page 17
- “_NBPE_PKT_READY_INFO Structure” on page 77

NBPE_Reset Function

Use this function to reset the Policy Accelerator.

```
VOID NBPE_Reset ( IN NBPE_DEV_HANDLE hDev);
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_REGISTER. See “NBPE_DEV_HANDLE Type” on page 74.

Description

The NIC driver can use this function to reset the Policy Accelerator (Processing Engine) and go back to the initial state. Both on-going transmitting and receiving packets to the NIC device are aborted. Moreover, all Policy Accelerator buffers regardless of their states are given back to Policy Accelerator.

See Also

- “Initiating Communication With the Policy Accelerator” on page 12

NBPE_ReturnXmitPEBuffer Function

Use this function to return transmitted buffers to the Policy Accelerator.

```
NBPE_STATUS NBPE_ReturnXmitPEBuffer (
          IN NBPE_DEV_HANDLE hDev,
          IN UINT32           numBuf,
          IN PHY_ADDR        *pXmitBuf );
```

Argument	Description
hDev	The handle for the current connection between the NIC driver and the PA-100 driver as initialized by IOCTL_NBOOST_FOREIGN_REGISTER. See “NBPE_DEV_HANDLE Type” on page 74.
numBuf	
pXmitBuf	

Description

Your NIC driver must use this function to return the transmitted buffers to thePolicy Accelerator.

If NBFIF_BoundTxPacketsReady is used, there is a fast return mechanism available for the bounded instance. When the NIC driver calls this function with numBuf set to 0 and pXmitBuf set to NULL, this function returns *all* buffers in the transmit buffer ring to the Policy Accelerator. When only pXmitBuf is NULL, this function treats the numBuf of buffers returned to be buffers returned sequentially on the transmit buffer ring and acts accordingly.

See Also

- “Handling Packets” on page 17

Index

O

P

R

S

T

WXYZ

IX SDK Software Developer's Kit License Agreement

IMPORTANT: You (the "licensee") are consenting to be bound by this agreement if you do any of the following:

- **Click on the "accept" button**
- **Install or use the software**
- **Otherwise exercise any rights provided below to use the accompanying Intel™ IX API Software Developer's Kit (the "Intel SDK")**

Or, if applicable, you are bound by a currently effective written agreement regarding the use of the Intel SDK and signed by an authorized agent of you and by an officer of Intel.

If you do not agree to the terms of this agreement or such signed agreement, as applicable, then do not use or copy the Intel SDK, and contact the place from which you obtained it, if any of these terms are considered an offer, acceptance is expressly limited to these terms.

This Agreement sets forth the terms and conditions of your use of the accompanying Intel SDK, together with documentation provided to you by Intel. Any third party software that is provided with the Intel SDK with such third party's license agreement (in either electronic or printed form), and your use of such third party software, shall be governed by such third party's license agreement in addition to this Agreement. As used in this Agreement, Intel shall mean Intel Corporation, its affiliates, or its subsidiaries.

Users of the Intel SDK pursuant to this Agreement must either be individuals using the license on their own behalf or be employees or contractors of a corporation or other entity which has accepted the terms of this Agreement and on behalf of which the Intel SDK is being used, in which case the term "Licensee" in this Agreement refers to you and such entity.

1. Grant of License.

a. Subject to the terms of this Agreement, Intel grants to Licensee a worldwide, nonexclusive, nontransferable, nonassignable, nonsublicensable license (the "License") under Intel's copyrights to (i) copy the Intel SDK and associated documentation for internal use to integrate Applications for use with Intel Products, and (ii) to make and distribute as many copies of the integrated applications containing the Intel SDK as necessary. "Intel Products" means approved Intel Hardware listed in the datasheet provided with this Intel SDK. "Applications" means Licensee's current and future expected applications that will use the Intel SDK.

b. Accompanying the Intel SDK is specific source code ("Intel Source") such as ARM Compiler, ARM Debugger, Include Files, and reference applications that Licensee may incorporate into Applications during the integration process using the Intel SDK. Subject to the terms of this Agreement, Intel grants to Licensee a worldwide, nonexclusive copyright license to reproduce, distribute, and sublicense to third parties the Intel Source in Licensee's Applications for use with Intel Products. Licensee recognizes that when it uses the Intel SDK to create or compile Applications, a portion of the Intel SDK, the Intel Source, will be compiled and linked into or with the Applications.

2. Ownership of the Intel SDK. As between the parties, Intel retains title to and ownership of, and all proprietary rights with respect to, the Intel SDK, the Intel Source, and all copies and portions thereof, whether or not incorporated into or with other Software. The License does not constitute a sale of the Intel SDK, the Intel Source, or any portion or copy of it.

3. Restrictions; Licensee Obligations.

a. Any redistribution or duplication of any software, code, and or application derived from the Intel SDK shall require that the Intel InstallShield installation program be used for installation or Licensee agrees to incorporate the Intel file license.txt in its entirety into Licensee's install program. The Intel-provided file license.txt includes all relevant copyright notices, trademark notices, and any other notices. Except as specified in the applicable user documentation provided by Intel, Licensee shall not (and shall not allow any third party to) (i) decompile, disassemble, or otherwise reverse engineer or attempt to reconstruct or discover any source code or underlying ideas or algorithms of the Intel SDK by any means whatsoever, (ii) remove any product identification, copyright or other notices, (iii) retarget any Intel SDK to interoperate with products other than Intel Products, or (iv) provide, lease, lend, use for timesharing or service bureau purposes, or otherwise use or allow others to use the Intel SDK to or for the benefit of third parties.

Confidential information disclosed under this license agreement, including the existence and content of this Agreement, shall be considered "Confidential Information." Use and disclosure of such Confidential Information shall be governed by the terms of the Corporate Single Use Nondisclosure Agreement or other Nondisclosure Agreement, signed between the parties and incorporated into this Agreement by reference.

4. Termination of License for Cause. This agreement will remain in effect unless Intel terminates it due to a breach of its terms. Upon termination, Licensee will cease all use of the Intel SDK and promptly destroy or return to Intel all printed materials and copies of the Intel SDK and all portions thereof (whether or not modified or incorporated with or into other software) and so certify to Intel. Except for the License and except as otherwise expressly provided herein, the terms of this Agreement shall survive termination. Termination is not an exclusive remedy, and all other remedies will be available whether or not the License or the Agreement is terminated.

5. Limited Warranty and Disclaimer. **The Intel SDK is provided "as is" without warranty of any kind including, without limitation, any warranty of merchantability or fitness for a particular purpose or noninfringement. Further, Intel does not warrant, guarantee, or make any representations regarding the use, or the results of the use, of the Intel SDK or written materials in terms of correctness, accuracy, reliability, or otherwise.** Licensee understands that Intel is not responsible for and will have no liability for hardware, software, or other items or any services provided by any person or entity other than Intel.

6. Export Restrictions. Licensee agrees to fully comply with all applicable United States and EEC or other countries regulations and laws in effect now and hereinafter, including compliance with the U.S. Foreign Corrupt Practices Act and all export laws, restrictions, national security controls and regulations on the distribution or dissemination of Applications or Intel Products, technology, and information related to and/or exchanged under this Agreement. Licensee agrees not to export or reexport, or allow the export or reexport of the Intel SDK or any Intel Product, Intel Proprietary Information, or any direct product thereof in violation of any such restrictions, laws or regulations, or without all required licenses and proper authorizations, to Cuba, Libya, North Korea, Iran, Iraq, or Rwanda or to any Group D:1 or E:2 country (or national of such country) specified in the then current Supplement No. 1 to Part 740 of the U.S. Export Administration Regulations (or any successor supplement or regulations).

7. Government Contracts. The Intel SDK is provided with RESTRICTED RIGHTS. If Licensee is the Government or a Government contractor, use, duplication or disclosure by the Government is subject to the restrictions as set forth in subparagraph (c)(1)(ii) or the Rights in Technical Data and Computer Software Clause as DFARS 252.227-7013 and FAR 52.227-19, as applicable. Manufacturer is Intel Corporation, 1350 Villa Street, Mountain View, California 94041-1126.

8. Limitation of Remedies and Damages. **To the maximum extent allowed by law, Intel shall not be responsible or liable with respect to any subject matter of this agreement under any contract, negligence, strict liability, or other theory: (a) for loss or inaccuracy of data or cost of procurement of substitute goods, services or technology; (b) for any special, indirect, incidental, or consequential damages including, but not limited to, loss of profits; or (c) for any matter beyond its reasonable control.**

Distribution of the Intel SDK is also subject to the following limitations: Licensees (i) are solely responsible to your customers for any update or support obligation or other liability that may arise from the distribution, (ii) do not make any statement that your product is "certified," or that its performance is guaranteed, by Intel, (iii) do not use Intel's name or trademarks to market your product without written permission, (iv) shall prohibit disassembly and reverse engineering, and (v) shall indemnify, hold harmless, and defend Intel and its suppliers from and against any claims or lawsuits, including attorney's fees, that arise or result from your distribution of any product.

9. Transfer; Successors. Licensee shall not assign this agreement or any part of it except with Intel's prior written consent.

10. General. This Agreement shall be governed by and construed under the laws of the State of Delaware and the United States without regard to conflicts of laws provisions thereof and without regard to the United Nations Convention on Contracts for the International Sale of Goods. In any action or proceeding to enforce rights under this Agreement, the prevailing party shall be entitled to recover costs and attorneys' fees. If any provision of this Agreement is held by a court of competent jurisdiction to be illegal, invalid or unenforceable, that provision shall be limited or eliminated to the minimum extent necessary so that this Agreement shall otherwise remain in full force and effect and enforceable. No rights or licenses with respect to the Intel SDK or Intel Products are granted, other than those rights expressly and unambiguously granted in this Agreement. This Agreement constitutes the entire agreement between the parties relating to the subject matter hereof.

Copyrights and Trademark Notification

Intel Corporation
1350 Villa Street
Mountain View, CA 94041-1126
Tel: 650.567.9800
Fax: 650.567.9810
www.netboost.com

Revision 1.0, May 2000