Uniform Driver Interface

UDI NIC Driver Specification Version 1.01

http://www.project-UDI.org/specs.html

UDI NIC Driver Specification

Abstract

The UDI NIC Driver Specification defines the required interfaces and semantics for UDI environments that support Network Interface Card (NIC) drivers. This is an optional extension to the UDI Core Specification, which is defined in a separate book. See the Document Organization chapter in the UDI Core Specification for a description of the other books in the UDI Specifications, as well as references to additional tutorial materials. The intended audience for this book includes driver writers, environment implementors, and metalanguage implementors.

Status of This Document

This document has been reviewed by Project UDI Members and other interested parties and has been endorsed as a Final Specification. It is a stable document and may be used as reference material or cited as a normative reference from another document. This version of the specification is intended to be ready for use in product design and implementation. Every attempt has been made to ensure a consistent and implementable specification. Implementations should ensure compliance with this version.

Copyright Notice

Acknowledgements

The authors would like to thank everyone who reviewed working drafts of the specification and submitted suggestions and corrections.

The authors would especially like to thank their significant others for putting up with the many hours of overtime put into the development of this specification over long periods.

Thanks to the following folks who contributed significant amounts of time, ideas, or authoring in support of the development of this specification or in working on the prototype implementations which helped us validate the specification:

Richard Arndt (IBM)
Bob Barned (Lockheed Martin)
Mark Bradley (Adaptec)
Darren Busing (Adaptec)
Steve Bytnar (STG)
Thomas Clark (Sun)
Deven Corzine
Jack Craig (SCO)
Betty Dall (HP)
Tim Damron (IBM)
Burkhard Daniel (STG)
Don Dugger (Intel)
Mark Evenson (HP)
Barry Feild (SCO)
Scott Feldman (Intel)
Mike Firman (STG)
Kurt Gollhardt (SCO)
Bob Goudreau (Data General)
James Hall (SCO/Sun)
Jim Heidbrink (Lockheed Martin)
Chris Herzog (STG)
Chris Ilnicki (HP)
Bret Indrelee (SBS Technologies)
David Kahn (Sun)
Matt Kaufmann (SCO)
Andrew Knutsen (SCO)
Ahuva Kroizer (Intel)
Man Fai Lau (SCO)
John Lee (Sun)
Robert Lipe (SCO)
Mike Lyons (IBM)
Alex Malone (DEC)
Lynne McCue (IBM)
Bill Nicholls
Guru Pangal (Starcom)
Mark Parenti (DEC)
James Partridge (IBM)
Scott Popp (SCO)
Hiremane Radhakrishna (Intel)
John Ronciak (Intel)
Kevin Quick (Interphase)
Larry Robinson (Adaptec)
Andrew Schweig (STG)
Sam Shteingart (HP)
Ajmer Singh (SCO)
James Smart (Compaq)
Pete Smoot (HP)
David Stoft (HP)
Rob Tarte (Pacific Codeworks)
Wolfgang Thaler (Sun)
Ramaswamy Tummala (Starcom)
Linda Wang (Sun)
Kevin Van Maren (Unisys)
Mike Wenzel (HP)

Countless people have helped in one way or another and any omissions or errors on our part in the list above are just that: omissions or errors on our part.

Thanks to Kevin Quick and the folks at Interphase for hosting the Interoperability events which have provided a great venue for validating both prototype and production UDI products.

Finally, thanks to David Roberts (Certek Software Designs) for designing the Project UDI logo.

Table of Contents

List of Reference Pages

Chapter 3 Network Interface Metalanguage

Alphabetical List of Symbols

NIC Driver Introduction 1

1.1 Introduction

This specification details the channel operations, parameters and sequences for the UDI Network Interface Metalanguage[1] which is used to support Network Interface Adapter Cards (*NIC*'s).

Each subsection defines the channel operation calls, control structure type declarations, the rationale for the operation's existence, constraints and guidelines for the use of each operation, and error conditions that can occur.

1.2 Scope

The networking device driver framework in UDI defines a set of channel operations and rules which allow for writing a NIC driver that works with existing and future networking protocol stacks regardless of the OS and protocol stack characteristics. This specification describes both the UDI *NIC Driver* (ND) and the UDI *Network Service Requester* (NSR) which communicate via the Network Interface Metalanguage. The ND is completely portable within the UDI environment and is intended to accompany the distribution of the NIC hardware. The NSR is typically provided by the UDI environment implementation or via additional UDI modules. The Network Interface Metalanguage completely defines the interface between the two components.

The UDI Network Interface Metalanguage was developed to represent a universal set of connectionless network-related functions that provide all of the needed functionality in an OS, protocol, and transport independent manner.

1.3 Normative References

The UDI Network Interface Metalanguage is designed to support a wide variety of network topologies. Although there is no single standard or reference for this implementation, the designer is expected to be familiar with the appropriate network protocol standards and technology specifications that apply to the implementation being developed. This includes International Standards Organization (ISO), American National Standards Inc. (ANSI), and Internet Engineering Task Force (IETF) publications for general networking standards as well as technology-specific standards from specific groups such as the ATM Forum, the ANSI X3T11 Fibre Channel committee, the various 802.x standards committees for Ethernet and FDDI, and other organizations and specifications as appropriate.

The UDI NIC Driver Specification also references and depends upon the UDI Core Specification.

1. While the term "Network Interface Metalanguage" is the proper name for this metalanguage, the term "NIC Metalanguage" is also used for brevity and is an equivalent reference.

1.4 Conformance

A conformant UDI NIC driver (ND) must fully implement the parent interface defined in this specification and must also be fully compliant with the UDI Core Specification interfaces. Other optional UDI interfaces may be supported if they are also conformant to the UDI Core Specification (0x101). No non-conformant interfaces may be used in a conformant UDI NIC driver.

A conformant UDI Network Service Requestor (NSR) implementing the interface to one or more network protocol stacks must fully implement the child interface defined in this specification and must also be fully compliant with the UDI Core Specification interfaces. Other optional UDI interfaces may be supported if they are also conformant to the UDI Core Specification (0x101). No non-conformant interfaces may be used in a conformant UDI NSR Module.

A conformant UDI Network Mapper must fully implement the child interface defined in this specification; other interfaces and implementation details are not constrained by Mapper conformance requirements.

1.5 Terminology

Throughout this document, the following conventions will be used:

- Operations of type "_req" are used to request an activity.
- Operations of type "_ack" are used to acknowledge the request (but not necessarily the success or failure in implementing that request).
- Operations of type "_ind" are used for indications of an event, typically asynchronous in nature.
- Operations of type "_rdy" are used to indicate that the corresponding driver is ready for an event from the other end of the channel.

The following terms will be used throughout the UDI NIC Driver Specification with the following intended meanings:

ND	Refers to the UDI Network Interface Adapter Card (NIC) driver supplied by the adapter manufacturer (typically) and which is used to manage and operate that adapter.
NSR	Refers to the UDI Network Service Requestor which is the child of the ND and implements the interface between the ND and the various network protocol stacks in the current operating environment.
NIC	Network Interface Card. This is the network adapter that is being controlled by the ND.
MAC	Media Access Controller. This customarily refers to the chip that is used to access the physical network media. Most of the functionality of a NIC card is represented by the MAC, which implements framing, physical signaling, and other operations to provide programmatic access to the network.
MAC Address	A Media Access Address which is typically the "hardware" or line-level address of a network node. This is customarily the address that the NIC uses to send packets on the network; protocol-specific virtual addresses (*e.g.* the IP address) are mapped to MAC addresses on transmit and recieve operations.

Unicast	The operational mode of the NIC whereby it receives packets which contain its MAC address as the packet destination address.
Multicast	The operational mode of the NIC whereby it recieves packets which contain either its own MAC address or a MAC address which matches one or more multicast addresses. It is intended that only one adapter shall receive and respond to unicast packets whereas multiple adapters shall receive and respond to multicast packets.
Broadcast	The operational mode of the NIC whereby all packets that can be received on the network connection are passed to the NSR, regardless of the destination address of those packets. Also known as “promiscuous mode”.
Packet	The fundamental block of data and associated network headers that is sent or received by the NIC. Each packet is expected to have a network header which minimally includes the destination MAC address followed by zero or more bytes of payload data. Note that some network technologies may further subdivide a packet into smaller transmission units (e.g. ATM cells) but this segmentation and reassembly activity should be invisible to the interface between the ND and the NSR.
QoS	Quality of Service. A term which describes various means of identifying and distinguishing between network traffic that is exchanged between two network nodes. This term is generic and its use is usually specific to the network media and protocol types and may include concepts such as: priority, bandwidth, error rate or recovery, and packet acknowledgements.

NIC Driver Requirements and Bindings 2

2.1 General Requirements

2.1.1 Versioning

All functions and structures defined in the UDI NIC Driver Specification are part of the "`udi_nic`" interface, currently at version "`0x101`". A driver which conforms to and uses the UDI Network Interface Metalanguage, Version 1.01, must include the following declaration in its `udiprops.txt` file (see Chapter 30, *"Static Driver Properties"*, of the UDI Core Specification):

```
requires udi_nic 0x101
```

In each UDI NIC driver source file, before including any UDI header files, the driver must define the preprocessor symbol, `UDI_NIC_VERSION`, to indicate the version of the UDI Network Interface Metalanguage to which it conforms, which must be the same as the interface version defined above.

```
#define UDI_NIC_VERSION 0x101
```

A portable implementation of the Network Interface Metalanguage must include a corresponding "provides" declaration in its `udiprops.txt` file, must define `UDI_NIC_VERSION`, and must conform to the requirements specified in the Metalanguage-to-Environment (MEI) interface defined in Chapter 27, *"Introduction to MEI"*, of the UDI Core Specification and Chapter 28, *"Metalanguage-to-Environment Interface"*, of the UDI Core Specification.

As defined in Section 30.4.6, "Requires Declaration," on page 30-6 of the UDI Core Specification, the two least-significant digits of the interface version represent the minor number; the remaining hex digits represent the major number. Versions that have the same "major version number" as an earlier version shall be backward compatible with that earlier version (*i.e.*, a strict superset).[1]

2.1.2 Header Files

Each UDI NIC driver source file must include the file "`udi_nic.h`" after it includes "`udi.h`", as follows:

```
#include <udi.h>
#include <udi_nic.h>
```

The "`udi_nic.h`" header file contains function prototypes and other definitions needed to use the UDI NIC interfaces.

1. As an exception to this version compatibility, version 1.0 (0x100) is not forward compatible with any other versions bearing the major number of 1; version 1.0 of the specification cannot be wholly implemented as a functional product.

2.2 NIC Metalanguage Model

The Network Interface Metalanguage is designed to support the interface between a Network Interface Card Driver (ND) and the associated network protocol stacks in the operating system. Under the UDI model, the NIC driver is primarily concerned with the management of the hardware itself, the state of the link, and with sending and receiving network packets. Management of the network address space, topology discovery, and packet composition/decomposition operations are not the responsibility of the NIC driver and are left to the network protocol stacks.

The NIC driver will bind to a single network protocol module (NSR); there is expected to be a one-to-one relationship between NIC driver instances and network protocol module instances with a single Network Interface Metalanguage binding between each pair. Any packet multiplexing/demultiplexing operations on the basis of protocol type, destination address, or other filter are the responsibility of the network protocol module and are not handled in the ND. All packets received by the NIC (subject to local address and multicast address filtering) are passed to the network protocol module for handling.

The NIC driver implements inbound and outbound flow control between itself and the protocol stack by controlling the acknowledgement and corresponding control block recycling across the NIC Metalanguage data channels. Each control block represents either a packet (received or to be transmitted) or a capability to receive or transmit a packet; when no capability is present the corresponding operation is "blocked", thereby imposing flow control characteristics on the corresponding modules.

Because the NIC Metalanguage model is based on an asynchronous as-needed operational basis none of the operations defined by this metalanguage are abortable with `udi_channel_op_abort` or recoverable (as described in Section 4.10.1, "Overview of Region-Kill," on page 4-6 of the UDI Core Specification).

2.3 Bindings to the UDI Core Specification

2.3.1 Instance Attributes Bindings

In each of the attribute tables below, the **ATTRIBUTE NAME** is a null-terminated string (see Section 15.2, "Instance Attribute Names," on page 15-1 of the UDI Core Specification); the TYPE column specifies an attribute data type as defined in **udi_instance_attr_type_t** on page 15-7 of the UDI Core Specification; and the SIZE column specifies the valid size range, in bytes, for each attribute.

2.3.1.1 Enumeration Attributes

The UDI Network Adapter Driver enumerates network access points provided by that adapter. The following table specifies the enumeration attributes that can be specified for each network access point enumerated:

Table 2-1 NIC Enumeration Attributes

ATTRIBUTE NAME	TYPE	SIZE	Description
`if_num`	`UDI_ATTR_UBIT32`	4	Instance number for the media interface port relative to the other interface ports on this device.

Table 2-1 NIC Enumeration Attributes

ATTRIBUTE NAME	TYPE	SIZE	Description
`if_media`	UDI_ATTR_STRING	1..8	Media type string as defined in Table 2-2 on page 2-3. The size range specified includes the null-terminator
`identifier`	`UDI_ATTR_STRING`	1..41	String representation of the Factory MAC address of the NIC media interface port (see ***`mac_addr`*** in **udi_nic_bind_cb_t** on page 3-23), as an uppercase UDI_ATTR_ARRAY8 encoding per Table 30-1 on page 30-16 of the UDI Core Specification. The size range specified includes the null-terminator.
`address_locator`	`UDI_ATTR_STRING`	1..11	String representation of `if_num`, as an ASCII-encoded decimal string. The size range specified includes the null-terminator.
`physical_locator`	`UDI_ATTR_STRING`	1..11	String representation of `if_num`, as an ASCII-encoded decimal string. The size range specified includes the null-terminator
`physical_label`	`UDI_ATTR_STRING`	1..64	Driver-defined string uniquely identifying this device (optional).

The following table defines the media string prefixes for the different media types to be specified in the locator attribute:

Table 2-2 Media Type Strings

Media Type	Media String
UDI_NIC_ETHER	“eth”
UDI_NIC_TOKEN	“tr”
UDI_NIC_FASTETHER	“fe”
UDI_NIC_GIGETHER	“ge”
UDI_NIC_VGANYLAN	“vg”
UDI_NIC_FDDI	“fddi”
UDI_NIC_ATM	“atm”
UDI_NIC_FC	“fc”
UDI_NIC_MISCMEDIA	“net”

2.3.1.2 Enumeration Attribute Ranking

To support the ranking of enumerated devices against available drivers for the `udi_mei_enumerate_rank_func_t`, each rankable enumeration attribute is assigned a numeric ranking value; the ultimate rank returned by the ranking function is the sum of the individual attribute ranking values for all attributes which are matched in the enumeration.

The rankable enumeration attributes and their ranking values are shown in the following table:

Table 2-3 NIC Enumeration Attribute Ranking Values

Enumeration Attribute	Ranking Value
if_num	8
if_media	16
identifier	32
physical_label	64

2.3.1.3 Filter Attributes

The `if_media` enumeration attribute may be used as a filter attribute. There is no range specification allowed for this filter; an exact value match must be made if this attribute is used for filtering.

2.3.1.4 Custom Attributes

The following well-known attributes must be included, if applicable, as “custom” declarations in the NIC driver’s `udiprops.txt` static driver properties file for appropriate configuration of the target driver:

Table 2-4 NIC Custom Attributes

ATTRIBUTE NAME	TYPE	SIZE	Description
`%speed_mbps`	`UDI_ATTR_UBIT32`	4	May be used to direct the adapter to use a specific link speed in MBps. This value may be used to represent high-speed network connections; for low speeds the `%speed_bps` attribute should be used. For example, for Fast Ethernet this would be set to 10 or 100 for 10Mbit or 100Mbit connections, respectively. If this attribute is not set or is set to zero, the speed shall be (auto) negotiated when the link is enabled.
`%speed_bps`	`UDI_ATTR_UBIT32`	4	This value is used in the same manner that the `%speed_mbps` attribute is used, although this value specifies a lower range of speeds. Drivers shall ignore `%speed_bps` if `%speed_mbps` is set and non-zero.

Table 2-4 NIC Custom Attributes

ATTRIBUTE NAME	TYPE	SIZE	Description
%duplex	UDI_ATTR_STRING	1..64	Must be set to "Full", "Half", or "Auto-negotiate" (if appropriate). The size range specified includes the null-terminator. If this attribute is not set, the duplex mode shall be auto-negotiated, or–if the device does not support auto-negotiation–set to the most typical setting for the media type.
%connector	UDI_ATTR_STRING	1..64	May be used to override auto-detection of the port's cable or connector type. The list of valid values is driver-defined but might include some of the following: "BNC", "AUI", "TP", "DB-9", "Copper GBIC", etc. The size range indicated includes the null-terminator for the string.

2.3.1.5 Parent-Visible Attributes

There are no defined parent-visible attributes for the Network Interface Metalanguage.

2.3.2 Trace Event Bindings

The following definitions describe usage of the trace events for the Network Interface Metalanguage and NIC Drivers. These trace events are defined in the "Trace Event Types" section of *"Tracing and Logging"* of the UDI Core Specification.

- UDI_TREVENT_IO_SCHEDULED
  - Trace indicating a new network transmit request is being handled.
- UDI_TREVENT_META_SPECIFIC_1
  - Trace indicating a network transmit request encountered an error.
- UDI_TREVENT_IO_COMPLETED
  - Trace indicating a network packet was received and is being handled.
- UDI_TREVENT_META_SPECIFIC_2
  - Trace indicating an incoming network packet has an error.

2.3.3 Static Driver Property Bindings

Some of the bindings for the static driver properties are defined in Section 2.1.1, "Versioning". This includes the definition of the relevant interface name(s) (*i.e.*, the <interface_name> parameter on the "requires" and "provides" and other property declarations), and the definition of the interface version number for this version of this specification.

The driver category to be used with the "category" declaration (see Section 30.5.3, "Category Declaration," on page 30-11 of the UDI Core Specification) by a portable implementation of the Network Interface Metalanguage Library shall be "Network Interface Cards".

2.3.4 Device Management Bindings

As described in Section 24.6.2, “Suspend,” on page 24-28 of the UDI Core Specification, a driver may be suspended to allow for physical or logical device modifications or replacement. Both the ND and the NSR may queue some amount of traffic but are expected to discard data transfer operations until subsequently resumed.

2.4 NIC Metalanguage State Diagram

See “Driver Instantiation” on page 24-2 of the UDI Core Specification for the general configuration sequence of UDI drivers. See “Management Metalanguage States” on page 24-37 of the UDI Core Specification for details on the Management Metalanguage states. The following state diagram shows the Network Interface Metalanguage state diagram (from the perspective of the ND) which is the set of states specific to the Network Interface Metalanguage.

Figure 2-1 NIC Metalanguage State Diagram

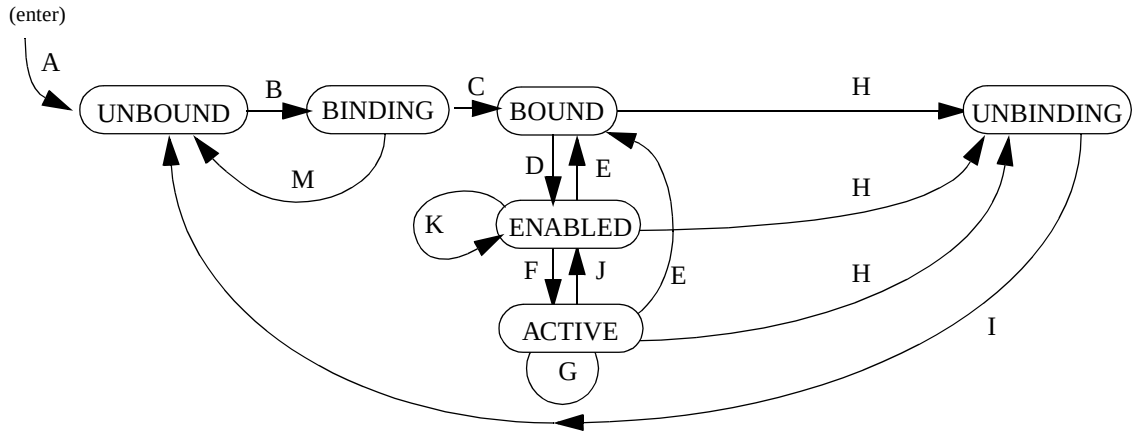

Table 2-5 NIC Metalanguage Events

Event	Operation
A	`udi_channel_event_ind` (`UDI_CHANNEL_BOUND`) *on NSR’s parent channel*
B	`udi_nd_bind_req`
C	`udi_nsr_bind_ack`
D	`udi_nd_enable_req` + `udi_nsr_enable_ack` with success
E	`udi_nd_disable_req`
F	`udi_nsr_status_ind` ***UDI_NIC_LINK_UP***
G	`udi_nsr_tx_rdy, udi_nd_tx_req, udi_nd_exp_tx_req, udi_nsr_rx_ind, udi_nsr_exp_rx_ind, udi_nd_rx_rdy`
H	`udi_nd_unbind_req`
I	`udi_nsr_unbind_ack`
J	`udi_nsr_status_ind` ***UDI_NIC_LINK_DOWN*** or ***UDI_NIC_LINK_RESET***
K	`udi_nsr_tx_rdy, udi_nd_tx_req, udi_nd_exp_tx_req`
M	`udi_nsr_bind_ack` with failure status

2.4.1 NIC Metalanguage States

UNBOUND — A network bind channel in the unbound state has been established between the two regions but has not yet been initialized in those regions for general use. The NSR (child) side of the network bind channel should initiate the network bind operation when in this state.

BINDING — This indicates that the NSR (child) side of the network bind channel has initiated a bind operation and is waiting for the ND (parent) side of the network channel to complete its initialization and acknowledge that bind request.

BOUND — This indicates that the network channels are fully bound between the two regions and that they are ready for other Network Interface Metalanguage operations. In the UNBOUND or BINDING states only the Network Interface Metalanguage bind operations should be used; once the BOUND state is reached the other metalanguage operations may be used.

ENABLED — This indicates that the ND has enabled the interface for I/O activity on the network. This does not necessarily indicate that the link is up.

ACTIVE — This indicates that the ND has enabled the interface and that the link is up. This is the only state where it is expected that network packet transmit and receive operations will be successful.

UNBINDING — This indicates that the Network channels are being shutdown. The NSR can cause this state to be entered by issuing a `udi_nd_unbind_req`. When the unbind operation is acknowledged then both the ND and the NSR will progress to the UNBOUND state.

Table 2-6 NIC Metalanguage: Valid Operations by State

Operation	UNBOUND	BINDING	BOUND	ENABLED	ACTIVE	UNBINDING
udi_nd_bind_req	**YES**	no	no	no	no	no
udi_nsr_bind_ack	no	**YES**	no	no	no	no
udi_nd_unbind_req	no	no	**YES**	**YES**	**YES**	no
udi_nsr_unbind_ack	no	no	no	no	no	**YES**
udi_nd_enable_req	no	no	**YES**	no	no	no
udi_nsr_enable_ack	no	no	no	**YES**	no	no
udi_nd_disable_req	no	no	no	**YES**	**YES**	no
udi_nsr_status_ind	no	no	no	**YES**	**YES**	no
udi_nd_ctrl_req	no	no	**YES**	**YES**	**YES**	no
udi_nsr_ctrl_ack	no	no	**YES**	**YES**	**YES**	no
udi_nd_info_req	no	no	**YES**	**YES**	**YES**	no
udi_nsr_info_ack	no	no	**YES**	**YES**	**YES**	no
udi_nsr_tx_rdy	no	no	no	**YES**	**YES**	no
udi_nd_tx_req	no	no	**YES**	**YES**	**YES**	no
udi_nd_exp_tx_req	no	no	no	**YES**	**YES**	no
udi_nsr_rx_ind	no	no	**YES**	no	**YES**	no
udi_nsr_exp_rx_ind	no	no	no	no	**YES**	no
udi_nd_rx_rdy	no	no	no	**YES**	**YES**	no

Network Interface Metalanguage 3

This specification details the channel operations, parameters and sequences for the UDI Network Interface Metalanguage which is used to support Network Interface (*NIC*) Adapters.

Each subsection defines the channel operation calls, control structure type declarations, the rationale for the operation's existence, constraints and guidelines for the use of each operation, and error conditions that can occur.

3.1 Overview

The networking device driver framework in UDI defines a set of channel operations and rules which allow for writing a NIC driver that works with existing and future networking protocol stacks regardless of the OS and protocol stack characteristics. This specification describes both the UDI *NIC Driver* (ND) and the UDI *Network Service Requester* (NSR) which communicate via the Network Interface Metalanguage. The ND is completely portable within the UDI environment and is intended to accompany the distribution of the NIC hardware. The NSR is typically provided by the UDI environment implementation or via additional UDI modules. The Network Interface Metalanguage completely defines the interface between the two components.

The UDI Network Interface Metalanguage was developed to represent a universal set of connectionless network-related functions that provide all of the needed functionality in an OS, protocol, and transport independent manner.

3.2 Network Interface Metalanguage Environment

The UDI Network Interface Metalanguage environment may be implemented in one of two possible configurations as shown in Figure 3-1: either UDI protocol support or a mapper to native OS protocol stacks. The implementation chosen and the associated details are determined by the provider of the UDI environment and do not affect the UDI NIC Driver implementation.

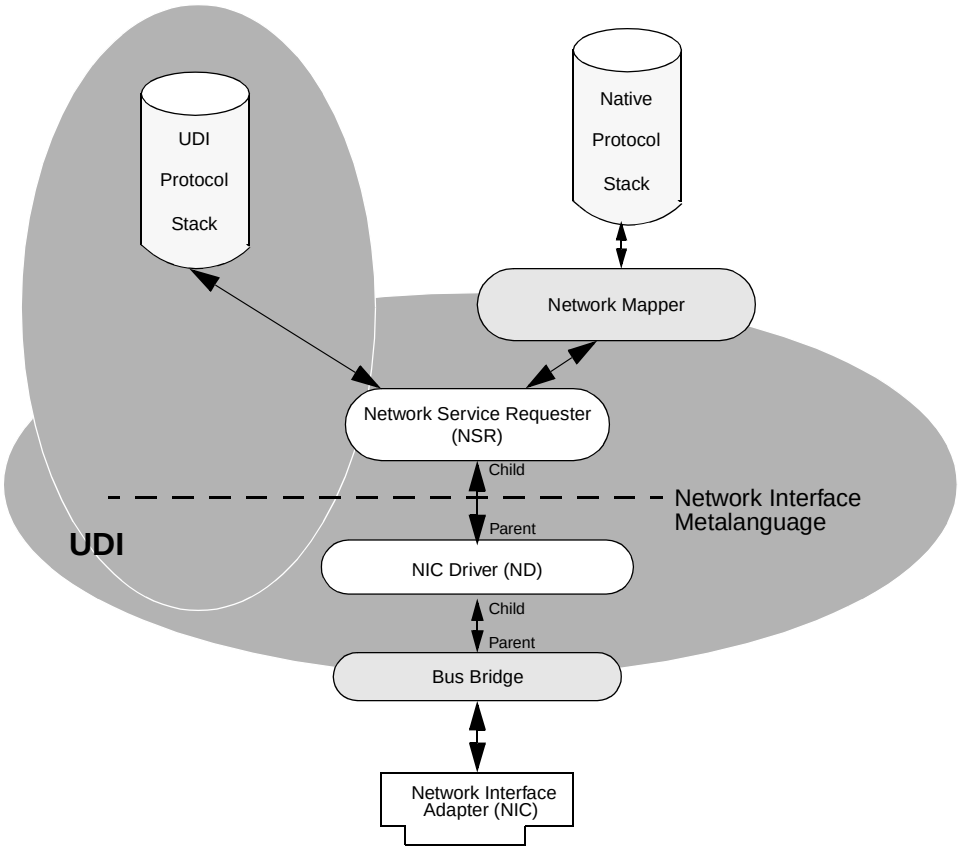

Figure 3-1 UDI Networking Environment

In the first configuration scenario, the protocol stack is implemented directly within the UDI environment. It is attached directly to the UDI ND in the customary UDI parent-child relationship. It implements the transport protocol mechanism using native UDI services (e.g. buffer services, timer services, etc.) and scheduling. The Protocol Stack operates as the Network Service requester (NSR) in this configuration, communicating with the ND via the Network Interface Metalanguage.

The second configuration scenario is used when the transport protocol mechanism is implemented in a manner native to that particular OS (e.g. Streams/DLPI, BSD ifnet/2, NDIS, ODI, CDLI, NDI, etc.). In this scenario the protocol stack is independent of UDI and does not run under the UDI paradigm and an additional component, the UDI Network Mapper is provided to act as the NSR agent on behalf of the native protocol stack. The UDI environment implementation must support the Network Interface Metalanguage by implementing one or the other of these support scenarios.

Please note that there are a number of “flow” diagrams presented throughout this chapter. They are intended to provide examples of how an existing Streams/DLPI protocol stack interface would be mapped to the corresponding Network Interface Metalanguage operations. The UDI Network Interface

Metalanguage can be implemented as an addition to or as a replacement for any existing network adapter interface, so the examples shown here should not be interpreted as the only possible implementation of this metalanguage.

3.2.1 Network Service Access

The UDI Network Adapter Metalanguage, as with many network driver architectures, defines that architecture in terms of a client-server arrangement. In terms of this arrangement, the ND provides various “services” to the protocol stack.

There may be multiple protocol stacks or other modules utilizing an ND driver at any single time, depending on the system configuration; these are all channeled through the NSR to provide a single access point between the ND and the NSR for all network traffic. Correspondingly, there may be multiple ND’s providing network access services to the protocol stack or stacks within the environment.

The services provided by a ND and their parameters are described by the UDI Network Interface Metalanguage. The way in which these services are used by the protocol stack is not described by the Metalanguage and is a prerogative of that protocol stack and accompanying environment.

One example of network access is an Ethernet datalink path between an Ethernet Adapter’s ND and the TCP/IP protocol stack via the NSR. TCP/IP packets are presented to the ND through Network Interface Metalanguage channel; the NSR will prepare the packet format for transmission and build the packet headers and the ND will perform the appropriate actions to post the buffer to the Adapter’s MAC chip to transmit it on the network. Incoming packets are verified, and the packet is passed to the NSR over the Network Interface Metalanguage channel. The NSR then interprets the frame header and removes it and the subsequent data buffer is passed to the TCP/IP protocol stack agent corresponding to the Ethertype in the Ethernet header.

3.2.2 Network Interface Multiplexing

As defined, the Network Interface Metalanguage provides an interface to the NIC driver that is nearly independent of the network protocol type. The NIC Driver’s primary responsibility is sending and receiving network packets via the associated hardware.

The NSR provides the network header creation and manipulation operations as well as multiplexing between the NIC and the various protocol stacks. The ND therefore can be implemented very simply since it only manages one hardware device and passes packets to one NSR.

In this paradigm, it is expected that an OS implementing multiple upper-layer protocols (e.g. IP, IPX, ARP, Appletalk, etc.) over a single network interface will attach these protocols to the NSR for header manipulation and incoming packet filtering and distribution. The NIC Driver will have only the single connection to the NSR for all of these simultaneous protocols. Accordingly, the only incoming packet filtering performed by the NIC Driver is to only pass up packets intended for the local host (unless broadcast/promiscuous mode is set, in which case ALL packets should be passed to the NSR).

Only one NSR may connect to a NIC Driver at any time. If a NIC Driver instance is bound via the Network Interface Metalanguage to a specific NSR child, no other Network Interface Metalanguage bind operations will be issued to that NIC Driver instance until the existing NSR child is unbound.

In addition, any Generic Metalanguage channels bound to a NIC Driver will be used in a manner defined by that NIC Driver and the associated client. However, it is customary for the NIC to refrain from passing any received packets to the Generic Metalanguage unless specifically requested via a Generic I/O Metalanguage control operation.

3.2.3 Network Addressing

The Network Service requester (NSR) is responsible for determining the network (MAC) destination address for transmit packets and for building the datalink header (*e.g.* Ethernet V2, 802.2, SNAP) in the packet before delivering the packet to the ND for transmission. On receives, the ND will validate any addressing information (if necessary) and then pass the entire packet to the NSR. The NSR will parse and remove the datalink header(s) from the packet and then pass it to the appropriate protocol stack entity, thereby providing the receive demultiplexing functionality.

When the Network Interface Metalanguage binding between the NSR and the ND is performed, the ND will indicate the media type to provide basic framing information to the NSR. The NSR will use this information to generate the proper device or technology headers for transmit operations and to correspondingly interpret the receive headers.

3.2.3.1 Address Specifications

During normal operations, it isn't necessary for the NSR and ND to exchange address information; the initial indication of the media type allows both the NSR and ND to parse the packet buffers to obtain or set MAC addresses as needed. In the event that it is necessary for the NSR and ND to exchange MAC addresses directly, they will use explicit fields in the associated operation control blocks to do so. Some examples of situations requiring the exchange of addresses are: ND reporting of current/factory MAC address to the stack, and NSR specification of multicast addresses to be accepted.

In order to allow the Network Interface Metalanguage to support a wide variety of media types, the size of the MAC addresses is not restricted to the traditional 6 bytes. Instead, the size of the MAC address will be explicitly stated when exchanging these addresses, subject to the maximum size of `UDI_NIC_MAC_ADDRESS_SIZE`[1]. Within the context of the ND, it may be reasonable to assume a MAC address length relating to the hardware in question and the ND may be written using this assumption, but in exchanges with the NSR, the size of the MAC address must be explicitly specified.

The current value of UDI_NIC_MAC_ADDRESS_SIZE is 20 bytes, which allows complete specification of E.164 addresses.

3.2.3.2 Address Implementations

This metalanguage specification uses the term "MAC" (Media Access Control) address to refer to the addresses which the ND and NSR will both evaluate. While this term is used extensively in conventional LAN implementations (e.g. Ethernet or FDDI), it usage is somewhat more vague in non-LAN or extended functionality protocols such as ATM or Fibre Channel. The contents of Table 3-1 on page 5 identifies media types and the recommended MAC address for those media. Media types not specifically identified in the table should interpret the MAC address to correspond to the static node identifying address that ARP would typically use to map an IP address. A NIC driver is not required to use the

1. The UDI_NIC_MAC_ADDRESS_SIZE specification is defined as part of the UDI NIC Driver Specification and may not be tailored by the individual UDI environment implementations. The value of this definition may only change as part of the evolution of this specification.

address values in the table below but it should be prepared to detect and process the contents of any address resolution protocol packets (*e.g.* ARP), in both directions, if it does not use the conventional addressing format.

Table 3-1 Media Types and MAC Address Lengths

Media Type	Description	MAC Address
UDI_NIC_ETHER	10 Mbit Ethernet	6-byte MAC address
UDI_NIC_TOKEN	Token-ring Media	6-byte MAC address
UDI_NIC_FASTETHER	100 Mbit Ethernet	6-byte MAC address
UDI_NIC_GIGETHER	1 Gbit Ethernet	6-byte MAC address
UDI_NIC_VGANYLAN	100 MBit 100VG-AnyLAN	6-byte MAC address
UDI_NIC_FDDI	100 Mbit FDDI	6-byte canonical MAC address
UDI_NIC_ATM	25 Mbit, 155 Mbit, or 622 Mbit ATM	Up-to 20-byte E.164 node address
UDI_NIC_FC	1 Gbit (or greater) Fibre Channel	8-byte WWN
UDI_NIC_MISCMEDIA	Unknown or unspecified media type	6-byte MAC address

3.2.4 Network Services

The network services defined by the UDI Network Interface Metalanguage specification which are provided by the ND may be divided into two areas: control operations and transfer operations. Each type of operation is handled by one or more UDI channels between the ND and the NSR. The UDI channels are specific to the type of operation associated with that channel (e.g. it is an error to send control operations over a transfer channel).

3.2.4.1 Control Operations

The control operations specified by the UDI Network Interface Metalanguage are used to specify the binding and to handle protocol-level control operations such as hardware MAC address registration, statistics, multicast addressing, etc.

3.2.4.2 Transfer Operations

The transfer operations handle simple datagram packet transmission and reception operations.

When the ND is provided with a packet buffer to transmit, it will also be given a buffer containing the data to transmit including any protocol and frame level headers. If the device does not require any special handling of the header information, the packet is simply queued to the device. If header translation or manipulation is required, the ND is free to perform these operations before queueing the packet for transmission. When the packet transmission has completed, the ND will indicate this completion to the NSR.

When the ND receives a packet from the line, it (or the NIC hardware on its behalf) will examine the link-level headers and trailers to determine both the source and destination addresses. The destination address will be verified as the link-level MAC address of the ND's interface or as an appropriate multicast or broadcast address (either perfect or hashed). Packets failing this verification will be discarded. The ND will then pass the packet to the NSR via a Network Interface Metalanguage receive

channel. The ND may chain multiple packets together for receive indications in a similar manner to the transmit packet chaining described above; the NSR may divide up the chain however it desires to acknowledge these receives in any sequence or sub-chain it desires.

The NSR is responsible for evaluating the link level header information to determine which local protocol stack agent the message should be directed to; if no local stack agent matches the link level addressing information, the NSR will discard the packet.

3.2.4.2.1 Transmit Flow Control

Transmit flow control is implemented by selective supply of transmit control blocks to the NSR by the ND. When the network connection is enabled, the ND will allocate a number of transmit control blocks and then provide these to the NSR. The NSR can then use these for transmit requests. The NSR may not internally allocate any additional transmit control blocks; it must wait for the ND to return transmit control blocks after transmission completes if it exhausts its supplied pool of control blocks.

The transmit control block also implements a completion urgency hint which is used to indicate to the ND the relative urgency of completing the transmit quickly. If not marked to indicate an urgency desire, the ND is free to complete the transmit at any point in the future that it desires. If urgency is indicated, it typically means that the packet buffer has been loaned to the networking subsystem by another OS module (e.g. NFS) and that the buffer should be released as soon as possible. This indicator allows the ND to optimize transmit completion interrupts as desired. Note that even if no urgency is indicated, the ND should insure that it quickly indicates a transmit completion if the NSR has passed all transmit control blocks to the ND, otherwise the flow control mechanism will prevent any further transmit requests.

To reduce overhead and allow batch optimizations in the ND, the NSR may pass a chain of multiple packets as part of a single Network Interface Metalanguage transmit request. Each packet should be transmitted in turn and when the transmission completes, the transmit buffer is deallocated and the transmission is acknowledged to the NSR. The ND may acknowledge on a packet-by-packet basis or on the basis of all (or part) of the original chain of packets. The driver is free to divide the chain into as many pieces as desired, as is the NSR or the UDI environment.

3.2.4.2.2 Receive Flow Control

Receive flow control is implemented in essentially the same manner as transmit flow control but in the reverse direction. The NSR will provide the ND with a supply of receive control blocks and associated buffers into which the received data is to be placed. Once a packet has been received into the buffer, the ND will pass the receive control block back to the NSR. The ND will never internally allocate receive control blocks and will instead wait for the NSR to supply them. As a result, the ND is not expected to retain any packets received while there are no available receive control blocks (although it may optionally queue them internally at its prerogative).

3.2.4.3 Transfer Channels

In order to allow separate handling of receive and transmit operations, the UDI Network Interface Metalanguage defines two separate transfer channels for the connection between the NSR and the ND: the transmit channel and the receive channel. Both channels may be handled by a single driver region or if desired they may each be handled by a separate region within the driver instance.

In addition, each channel provides channel operations for both normal and expedited data. This allows the NSR and ND to differentiate expedited, out-of-band, or other non-standard Quality of Servic (QoS) data from the normal in-band data being transferred. Protocols which support this model of data transfer (e.g. 100VG-AnyLAN) can use different channel operation routines to handle the different traffic types separately. For protocols which do not support this differentiation, the normal and expedited channel operations may point to the same routine for either the ND or the NSR.

3.2.5 Configuration Parameters

Different network technologies have associated parameters and modes of operation which are unique to that network technology (e.g. auto-negotiation for speed). Additionally, many network adapters have varying degrees of support for network-specific functionality in the hardware itself. To attempt to capture all of these variations in this metalanguage as part of the description of the communication between the NSR and the ND would be impossible and arbitrarily restrict current and future innovation. Additionally, this level of functionality is really only used for adapter hardware configuration and the results of this configuration are not interesting to nor do they affect the NSR and the general communications between the NSR and the ND.

Therefore, these types of adapter-specific configurations are not defined as part of this metalanguage but instead should be implemented as custom attribute declarations as part of the driver's static configuration properties (see the "Custom Declaration" section of *"Static Driver Properties"* in the UDI Core Specification). In this way, the configuration parameters can be defined on a per-driver basis with defaults and values provided by the manufacturer of the adapter as appropriate to that adapter. Example configuration parameters of this type are:

- Full or Half Duplex Configuration
- Link Speed setting (including auto-negotiation)
- Port Type selection (e.g. BNC, AUI, TP)

To change the any of these configuration parameters, the system administrator should update the values for these parameters in the system's persistent storage database to the newly desired values. The ND should re-scan the persistent storage database for any configuration parameter changes each time a link enable request is received, thereby allowing the system administrator to verify that the parameters have taken effect by (re)enabling the interface.

3.2.6 Hardware Checksum Offloads

Many hardware adapters provide the capability of calculating and/or verifying network packet checksums via offload capabilities of the hardware. This can significantly improve performance for networking operations and unlike the configuration parameters discussed above this capability does affect the operation of the NSR and its children. However, hardware checksum offloads are implemented in different manners and some adapters have different offloading capabilities than others.

To support hardware checksum offloading, the ND should utilize buffer tags[2] and the buffer tag assist functions: `udi_buf_tag_compute` and `udi_buf_tag_apply`.

2. See the the "Buffer Tags" section of *"Buffer Management"* in the UDI Core Specification for more information on buffer tags.

For a transmit operation, the NSR (or its children) will set various checksum set-request tags (e.g. `UDI_BUFTAG_SET_iBE16_CHECKSUM, UDI_BUFTAG_SET_TCP_CHECKSUM`, and `UDI_BUFTAG_SET_UDP_CHECKSUM)` for the data buffer to indicate what region of the buffer requires a checksum. If the hardware supports checksum offloading, it should use these tags to program the hardware accordingly. If the hardware does not support checksum offloading, the ND should use the `udi_buf_tag_apply` utility function to manually calculate and set the checksum(s) in the outgoing packet.

For received packets, if the hardware has performed checksum calculation on the incoming packet, that checksum value should be set via one or more `UDI_BUFTAG_BE16_CHECKSUM` tags on the received buffer to allow the NSR (or its children) to easily obtain the calculated buffer checksum. The NSR will use the `udi_buf_tag_compute` utility function to determine the checksum for received packets; the `udi_buf_tag_compute` utility will make use of any existing `UDI_BUFTAG_BE16_CHECKSUM` tags to avoid performing manual calculations, however, those tags are not required which effectively supports the case where the adapter hardware does not provide incoming packet checksum calculation.

If the hardware also (or instead) performs received packet checksum validation (i.e. it calculates the incoming packet's checksum as described above and verifies it against the checksum value(s) written by the transmitting host) it should indicate the success or failure of the validation via one or more of the status bits in the receive indication.[3]

Packets which have had this checksum validation performed may or may not additionally have the `UDI_BUFTAG_BE16_CHECKSUM` tag set but the ND should set this tag whenever the corresponding value is known if the NSR has indicated that it can utilize this value.

3. The use of buffer tags to indicate the good or bad checksum status is deprecated. The tags are still defined but all ND and NSR implementations should use the status bits in the receive control block instead. The tags will be removed in a future version of the UDI Specification. The specific tags that are deprecated are: `UDI_BUFTAG_TCP_CKSUM_GOOD`, `UDI_BUFTAG_TCP_CKSUM_BAD`, `UDI_BUFTAG_UDP_CKSUM_GOOD`, `UDI_BUFTAG_UDP_CKSUM_BAD`, `UDI_BUFTAG_IP_CKSUM_GOOD`, `UDI_BUFTAG_IP_CKSUM_BAD`.

3.3 Network Interface Management

This section describes the various control structures and operations that are used to configure the NIC Driver (ND) and the Network Service requester (NSR) and to establish a binding between these two entities. The Network Interface Management description can be further subdivided into the following groups:

- UDI Initialization and Registration
- Network Control Block Management
- NIC Metalanguage Binding
- Network Adapter Control

This section assumes that the reader is familiar with the UDI Management Metalanguage and the UDI configuration process.

3.3.1 Initialization and Registration Operations

The initialization and registration definitions are used to prepare the driver and environment for support of the Network Interface Metalanguage and the corresponding Network Adapter Driver (ND). These definitions are static for the entire Network Driver or NSR and relate to the configuration established for the driver by its `udiprops.txt` file (see Chapter 30, *"Static Driver Properties"*, of the UDI Core Specification).

3.3.2 Channel Ops Vector Registration

The channel operations vector registration is used to register the driver entry points for the parent driver (ND) and/or the child driver (NSR network interface mapper or protocol module). The ND performs registration by referencing `udi_ops_init_t` structures (via the `udi_init_info` declaration) for the following operations vectors:

- `udi_nd_ctrl_ops_t`
- `udi_nd_tx_ops_t`
- `udi_nd_rx_ops_t`

The NSR performs registration by declaring `udi_ops_init_t` structures for the following operations vectors:

- `udi_nsr_ctrl_ops_t`
- `udi_nsr_tx_ops_t`
- `udi_nsr_rx_ops_t`

The primary functions used for Network Interface Metalanguage channel creation and manipulation are specified by the `udi_nxx_ctrl_ops_t`. The primary channel created between the ND and the NSR will use these operations to establish the Network Interface Metalanguage data transfer channels between the ND and the NSR.

NAME **udi_nd_ctrl_ops_t** *ND control entry point ops vector*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

typedef const struct {
     udi_channel_event_ind_op_t *channel_event_ind_op;
     udi_nd_bind_req_op_t *nd_bind_req_op;
     udi_nd_unbind_req_op_t *nd_unbind_req_op;
     udi_nd_enable_req_op_t *nd_enable_req_op;
     udi_nd_disable_req_op_t *nd_disable_req_op;
     udi_nd_ctrl_req_op_t *nd_ctrl_req_op;
     udi_nd_info_req_op_t *nd_info_req_op;
} udi_nd_ctrl_ops_t;

/* ND Control Ops Vector Number */
#define UDI_ND_CTRL_OPS_NUM              1
```

DESCRIPTION A Network Driver uses the `udi_nd_ctrl_ops_t` structure in a `udi_ops_init_t` as part of its `udi_init_info` in order to register its entry points for the Network Interface Metalanguage control channel operations.

REFERENCES `udi_nd_tx_ops_t`, `udi_nd_rx_ops_t`

EXAMPLE

A typical Network Driver may include the following declarations:

```
/* Forward Declarations */
static udi_channel_event_ind_op_t
              my_channel_event_handler;
static udi_nd_bind_req_op_t my_bind_handler;
static udi_nd_unbind_req_op_t my_unbind_handler;
static udi_nd_enable_req_op_t my_enable_handler;
static udi_nd_disable_req_op_t my_disable_handler;
static udi_nd_ctrl_req_op_t my_control_req_handler;
static udi_nd_info_req_op_t my_info_provider;

static const udi_nd_ctrl_ops_t my_ctrl_ops = {
     my_channel_event_handler,
     my_bind_handler,
     my_unbind_handler,
     my_enable_handler,
     my_disable_handler,
     my_control_req_handler,
     my_info_provider
};

#define MY_ND_CTRL_OPS  4
#define MY_NET_META     1 /* meta_idx */


static const udi_ops_init_t my_net_ops_init[] = {
     {
          MY_ND_CTRL_OPS,
          MY_NET_META,
          UDI_ND_CTRL_OPS_NUM,
          sizeof(my_net_child_data_t),
          (udi_ops_vector_t *)&my_ctrl_ops,
     },
     ...
};
```

Which would then be used with the following `udiprops.txt` entries:

```
requires udi_nic 0x101
child_bind_ops 1 0 4 # Net Meta bind for primary rgn
```

NAME **udi_nd_tx_ops_t** *ND transmit entry point ops vector*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

typedef const struct {
     udi_channel_event_ind_op_t *channel_event_ind_op;
     udi_nd_tx_req_op_t *nd_tx_req_op;
     udi_nd_exp_tx_req_op_t *nd_exp_tx_req_op;
} udi_nd_tx_ops_t;

/* ND TX Ops Vector Number */
#define UDI_ND_TX_OPS_NUM                2
```

DESCRIPTION

A Network Driver uses the `udi_nd_tx_ops_t` structure in a `udi_ops_init_t` as part of its `udi_init_info` in order to register its entry points for the Network Interface Metalanguage transmit channel operations.

REFERENCES `udi_nd_ctrl_ops_t`, `udi_nd_rx_ops_t`

NAME

udi_nd_rx_ops_t *ND receive entry point ops vector*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

typedef const struct {
     udi_channel_event_ind_op_t *channel_event_ind_op;
     udi_nd_rx_rdy_op_t *nd_rx_rdy_op;
} udi_nd_rx_ops_t;

/* ND Receive Ops Vector Number */
#define UDI_ND_RX_OPS_NUM                3
```

DESCRIPTION

A Network Driver uses the `udi_nd_rx_ops_t` structure in a `udi_ops_init_t` as part of its `udi_init_info` in order to register its entry points for the Network Interface Metalanguage receive channel operations.

REFERENCES

`udi_nd_ctrl_ops_t`, `udi_nd_tx_ops_t`

NAME

udi_nsr_ctrl_ops_t *NSR control entry point ops vector*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

typedef const struct {
     udi_channel_event_ind_op_t *channel_event_ind_op;
     udi_nsr_bind_ack_op_t *nsr_bind_ack_op;
     udi_nsr_unbind_ack_op_t *nsr_unbind_ack_op;
     udi_nsr_enable_ack_op_t *nsr_enable_ack_op;
     udi_nsr_ctrl_ack_op_t *nsr_ctrl_ack_op;
     udi_nsr_info_ack_op_t *nsr_info_ack_op;
     udi_nsr_status_ind_op_t *nsr_status_ind_op;
} udi_nsr_ctrl_ops_t;

/* NSR Control Ops Vector Number */
#define UDI_NSR_CTRL_OPS_NUM              4
```

DESCRIPTION

A Network Service Requestor (NSR) uses the `udi_nsr_ctrl_ops_t` structure in a `udi_ops_init_t` as part of its `udi_init_info` in order to register its entry points for the Network Interface Metalanguage control channel operations.

REFERENCES

`udi_nd_ctrl_ops_t`, `udi_nsr_tx_ops_t`, `udi_nsr_rx_ops_t`

NAME

udi_nsr_tx_ops_t *NSR transmit entry point ops vector*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

typedef const struct {
     udi_channel_event_ind_op_t *channel_event_ind_op;
     udi_nsr_tx_rdy_op_t *nsr_tx_rdy_op;
} udi_nsr_tx_ops_t;

/* NSR Transmit Ops Vector Number */
#define UDI_NSR_TX_OPS_NUM                5
```

DESCRIPTION

A Network Service Requestor (NSR) uses the `udi_nsr_tx_ops_t` structure in a `udi_ops_init_t` as part of its `udi_init_info` in order to register its entry points for the Network Interface Metalanguage transmit channel operations.

REFERENCES

`udi_nsr_ctl_ops_t`, `udi_nsr_rx_ops_t`

NAME **udi_nsr_rx_ops_t** *NSR receive entry point ops vector*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

typedef const struct {
     udi_channel_event_ind_op_t *channel_event_ind_op;
     udi_nsr_rx_ind_op_t *nsr_rx_ind_op;
     udi_nsr_exp_rx_ind_op_t *nsr_exp_rx_ind_op;
} udi_nsr_rx_ops_t;

/* NSR Receive Ops Vector Number */
#define UDI_NSR_RX_OPS_NUM                6
```

DESCRIPTION

A Network Service Requestor (NSR) uses the `udi_nsr_rx_ops_t` structure in a `udi_ops_init_t` as part of its `udi_init_info` in order to register its entry points for the Network Interface Metalanguage receive channel operations.

REFERENCES `udi_nsr_ctrl_ops_t`, `udi_nsr_tx_ops_t`

3.3.3 Network Control Block Registration

The Network Interface Metalanguage uses a number of different types of control blocks for it's various metalanguage operations. These control blocks contain supplementary information for the operation being performed, scratch space for utilization by the owner of the control block, and environment management information for controlling each operation and marshalling arguments when necessary (the environment management portion of the control block is not visible to the ND).

The typical usage of these control blocks by an ND driver or NSR is as follows (from the ND perspective):

1. Register the control block index for each type of control block by filling in `udi_cb_init_t` structures referenced by the `udi_init_info` structure.
2. Register the ops index for each type of channel by filling in `udi_ops_init_t` structures referenced by the `udi_init_info` structure.
3. All subsequent channel operations will receive a control block appropriate to that operation type on the corresponding channel. The control block and its associated scratch space will be sized as specified by the registration in the previous steps. The ND should perform the requested operation and then return the control block in the corresponding acknowledgement channel operation.
4. When allocating a control block internally for use, the control block index argument passed to `udi_cb_alloc` specifies the appropriate type of control block; the resulting allocated control block will be large enough to be used for any of the control blocks within that index group.

3.3.4 Network Interface Metalanguage Binding

3.3.4.1 Network Bind Operation

Once the ND has been initialized and it has registered the UDI Network Interface Metalanguage channel operations, it may receive bind requests from the NSR. As the first stage of this binding, the UDI Management Agent (MA) will bind the child (NSR) to the parent (ND) using the techniques described in the UDI Management Metalanguage chapter. A typical binding can be summarized by the following procedure:

1. The MA creates the initial bind channel between the ND and the NSR. This channel is the network control channel and uses the UDI Network Interface Metalanguage `ctrl_ops` role.
2. The MA sends a `udi_usage_ind` to the new ND and NSR regions separately to allow them to initialize.
3. The MA sends `udi_channel_event_ind` operations of type `BIND_CHANNEL` as needed to initialize any secondary regions and internal bind channels for the ND and NSR.
4. The MA sends a `udi_channel_event_ind` operation of type `BIND_CHANNEL` to the NSR region.
5. The NSR prepares its end of the channel for use. The UDI Network Interface Metalanguage control channel is now established between the ND and the NSR.

At this point, the UDI Network Interface Metalanguage control channel is established between the ND and the NSR; the NSR may now initiate Network Bind operations to the ND.

The Network Bind operation (`udi_nd_bind_req`) exchanges basic information about the media type and supported packet information between the NSR and the ND, as well as identifying the transfer channel indices to be used for communication.

1. The NSR spawns the transmit channel, specifying that the spawned channel should use the transmit channel ops vector. This may be done in the primary region or in another region created specifically to handle transmit operations.
2. The NSR spawns the receive channel in a manner similar to the transmit channel.
3. The NSR issues a `udi_nd_bind_req` operation over the newly established bind channel to the ND. Part of the `udi_nd_bind_req` operation information specifies the indices of the spawned transfer channels.

The ND and the NSR have now agreed to communicate and have a common control channel which allows them to communicate directly without requiring assistance from the UDI MA. However, as noted in Section 3.2.4.3, “Transfer Channels”, the ND and NSR cannot yet perform data transfer operations; the data handling channels have not been established and will be created by subsequent operations.

1. The ND verifies the request, prepares itself for handling that NSR binding.
2. The ND should then call `udi_channel_spawn` with the spawn index indicated in the Network Bind control block to establish the transmit data channel to the NSR. Simultaneously or on completion of that spawn it should also call `udi_channel_spawn` to establish the receive data channel to the NSR.

3. The ND then responds to the NSR with a `udi_nsr_bind_ack` operation over the network control channel. This acknowledgment should not be generated by the ND until the channel spawn operations have completed to ensure that the NSR does not issue operations to these channels before they are fully created.

4. The NSR then completes its internal preparations and then responds to the UDI MA over the management channel with a `udi_channel_event_complete` to indicate completion of the bind operation.

The expedited data ops vectors allow expedited data to be handled separately from normal data and thereby allows a higher priority to be assigned to managing expedited data. If no expedited data is expected, or if the network type or protocol stack cannot support expedited data, the expedited data ops vectors should point to the normal data handling routines.

The data channels are established by spawning the network control channel as indicated by the contents of the network bind control block exchanged in Step 3 above.

At this point, the communications pathways between the ND and the NSR have been fully established. The NSR may now enable the interface, which causes the ND to enable the hardware and to create a pool of transmit control blocks to post to the NSR for sending or receiving network packets. The normal flow of calls associated with the Network Bind operation which supports expedited data operations is illustrated in Figure 3-2.

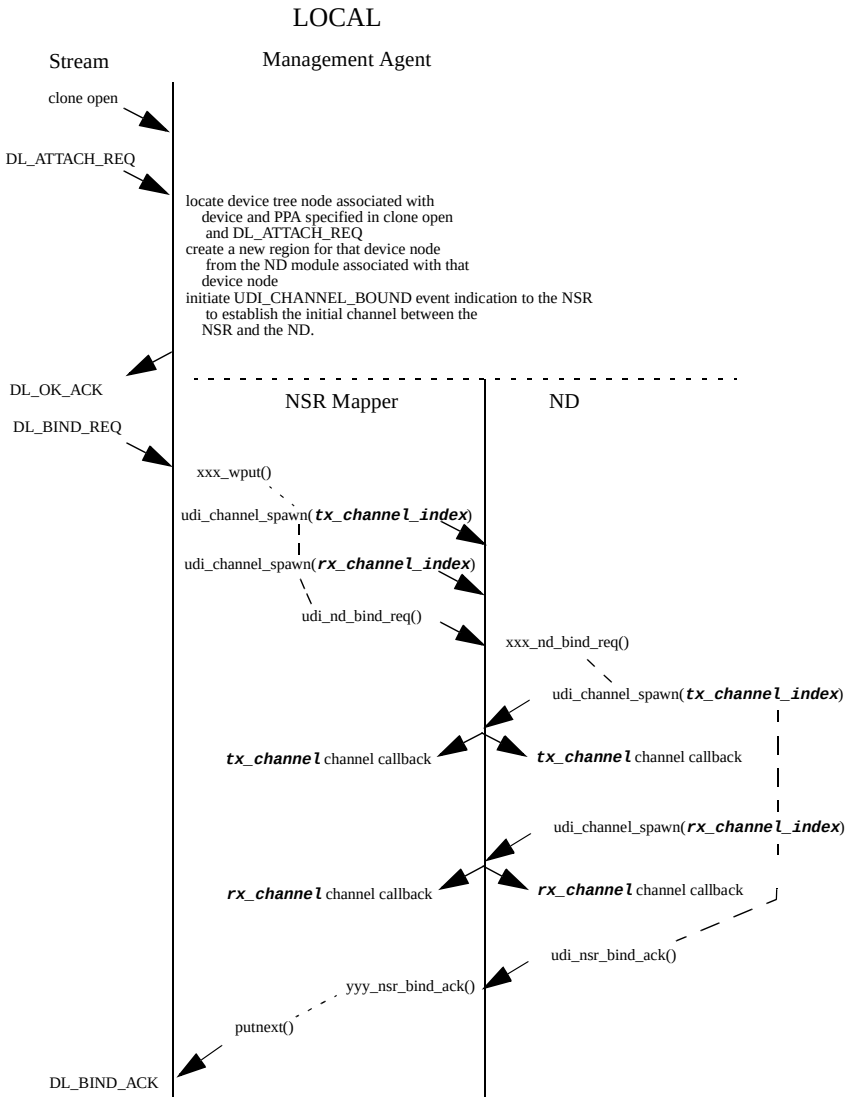

Figure 3-2 Connectionless Network Bind Operation

3.3.4.2 Network Unbind Operation

The Network Unbind process is used to detach and release the resources established by the Network Bind operation. During the Network Unbind operation the driver is instructed to stop forwarding any frames and shutdown the channels. The driver acknowledges the Network Unbind operation through a `udi_nsr_unbind_ack` operation. Once the network unbind has been performed, the control channel and both data channels should be released by calling `udi_channel_close.` Either end may initiate the closure, but both sides must release the data channels after the Network Unbind acknowledgement.

Note – One or both of the ND and NSR may receive `UDI_CHANNEL_CLOSED` indications depending on the sequence of events; these are assumed to be indications of the proper unbinding operation and are typically ignored.

The Network Unbind process is used to gracefully terminate network connectivity. If the normal data channels or the control channel is closed by either end, a Network Unbind condition is assumed and the channels are shutdown by both the NSR and the ND.

The normal flow of this operation is illustrated in Figure 3-3.

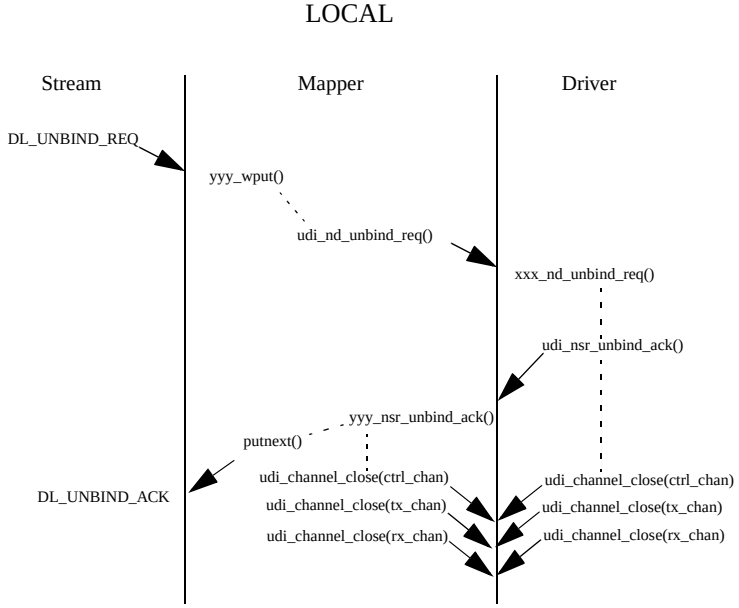

Figure 3-3 Unbind Operation

NAME **udi_nic_cb_t** *Standard network control block*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

typedef struct {
     udi_cb_t gcb;
} udi_nic_cb_t;

/* Network Standard Control Block Group Number */
#define UDI_NIC_STD_CB_NUM                1
```

MEMBERS

gcb is the generic control block header which includes a pointer to the scratch space associated with this block and the channel context for the associated channel. The driver may use the scratch space while it owns the control block, but the values are not guaranteed to persist across channel operations.

DESCRIPTION

The `udi_nic_cb_t` structure is used for channel operations between a child (NSR) and its parent (ND) where there is no additional metalanguage information needed in the control block. A number of the Network Interface Metalanguage control operations are defined to use this standard control block.

This control block must be declared by specifying the control block index value `UDI_NIC_STD_CB_NUM` in a `udi_cb_init_t` in the driver's `udi_init_info`.

The NSR or ND obtains the `udi_nic_cb_t` structure to use with Network Interface Metalanguage control operations by calling `udi_cb_alloc` with a ***`cb_idx`*** that has been defined for the `UDI_NIC_STD_CB_NUM` control block.

REFERENCES `udi_init_info, udi_cb_init_t, udi_cb_alloc`

NAME

udi_nic_bind_cb_t *Network bind control block*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

typedef struct {
     udi_cb_t gcb;
     udi_ubit8_t media_type;
     udi_ubit32_t min_pdu_size;
     udi_ubit32_t max_pdu_size;
     udi_ubit32_t rx_hw_threshold;
     udi_ubit32_t capabilities;
     udi_ubit8_t max_perfect_multicast;
     udi_ubit8_t max_total_multicast;
     udi_ubit8_t mac_addr_len;
     udi_ubit8_t mac_addr[UDI_NIC_MAC_ADDRESS_SIZE];
} udi_nic_bind_cb_t;

/* Network Bind Control Block Group Number */
#define UDI_NIC_BIND_CB_NUM              2

/* Maximum MAC Address Size Definition */
#define UDI_NIC_MAC_ADDRESS_SIZE         20

/* media_type values */
#define UDI_NIC_ETHER                    0
#define UDI_NIC_TOKEN                    1
#define UDI_NIC_FASTETHER                2
#define UDI_NIC_GIGETHER                 3
#define UDI_NIC_VGANYLAN                 4
#define UDI_NIC_FDDI                     5
#define UDI_NIC_ATM                      6
#define UDI_NIC_FC                       7
#define UDI_NIC_MISCMEDIA                0xff

/* capabilities indications */
#define UDI_NIC_CAP_TX_IP_CKSUM          (1U<<0)
#define UDI_NIC_CAP_TX_TCP_CKSUM         (1U<<1)
#define UDI_NIC_CAP_TX_UDP_CKSUM         (1U<<2)
#define UDI_NIC_CAP_MCAST_LOOPBK         (1U<<3)
#define UDI_NIC_CAP_BCAST_LOOPBK         (1U<<4)

/* capabilities requests */
#define UDI_NIC_CAP_USE_TX_CKSUM         (1U<<30)
#define UDI_NIC_CAP_USE_RX_CKSUM         (1U<<31)
```

MEMBERS

gcb is the generic control block header which includes a pointer to the scratch space associated with this block and the channel context for the associated channel. The driver may use the scratch space while it owns the control block, but the values are not guaranteed to persist across channel operations.

media_type is the network media type supported by the ND and associated Adapter hardware. This information is used by the NSR to parse and create frame headers for packets. Valid values for ***media_type*** are specified in Table 3-1 on page 5.

The ***media_type*** specified in this control block is expected to be the same as the `media_type` enumeration attribute value; where the values differ the value from this control block will be used. The same is true for the ***mac_addr*** specification.

If there is not an exact match for your media type appearing in the table, select the media type with the framing style closest to the device's network type. The ***media_type*** value suggests default values for the remaining `net_bind_ack_cb_t` configuration values and also hints to the NSR about handling packets for this network interface.

min_pdu_size is the minimum size of a data packet including the headers that the NSR or other modules will supply. The ND must return this information in the ack operation. If specified as 0, the default is used based on the specified ***media_type*** (as specified in Table 3-1).

max_pdu_size is the maximum size of a data packet including the headers that the NSR or other modules will supply. The ND must return this information in the ack operation. If specified as 0, the default is used based on the specified ***media_type***.

rx_hw_threshold is the number of receive requests that may be posted to the hardware adapter at any one time. This value is used as a hint to the NSR to indicate the number of receive control blocks and buffers that should be provided to the ND for receive operations. The ND should be prepared to operate with more or less than this value but this should represent the optimal requirements of the ND and its hardware adapter.

capabilities specifies the capabilities of the Network Driver and its associated hardware. This information will be used by the NSR and protocol stacks to determine how to most effectively utilize the hardware in the current environment.

`UDI_NIC_CAP_TX_IP_CKSUM` indicates that the hardware is capable of automatically detecting outbound IP packets and generating and inserting IP checksum values for those packets.

`UDI_NIC_CAP_TX_TCP_CKSUM` indicates that the hardware is capable of automatically detecting outbound TCP packets and generating and inserting TCP checksum values for those packets.

If this capability is expressed by the ND then the NSR and corresponding IP module must ensure that: either the entire

TCP packet is sent to this ND as a single IP fragment or else the NSR or IP module must calculate and insert the TCP checksum itself when the packet is fragmented. This is required because the NSR or IP module may choose to send different fragments of the request to different ND modules, therefore calculation of the TCP checksum in the ND is not possible (and many hardware adapters lack the capability of calculating a checksum across multiple fragments of a TCP packet).

The ND can assist in ensuring that the TCP packet is received in its entirety by indicating a large ***max_pdu_size*** and performing actual MTU fragmentation itself.

`UDI_NIC_CAP_TX_UDP_CKSUM` indicates that the hardware is capable of automatically detecting outbound UDI packets and generating and inserting UDP checksum values for those packets.

`UDI_NIC_CAP_MCAST_LOOPBK` indicates that the hardware will receive it's own transmitted multicast packets if the multicast address matches the hardware's multicast receive addressing. The NSR should use this information to determine how multicast loopback should be handled in the environment implementation.

`UDI_NIC_CAP_BCAST_LOOPBK` indicates that the hardware will receive it's own transmitted broadcast packets. The NSR should use this information to determine how broadcast loopback should be handled in the environment implementation.

In addition, the NSR may request various functionality from the Network Driver. Any capabilities requests of this type are simply requests and not requirements; the ND is not required to supply the requested functionality.

`UDI_NIC_CAP_USE_RX_CKSUM` indicates that the NSR will utilize any receive checksum buffer tags that are attached to the buffer by the ND. This flag is primarily used to optimize performance: for an NSR which will not utilize a receive checksum buffer tag, clearing this flag will reduce overhead in the ND associated with generating this tag.

`UDI_NIC_CAP_USE_TX_CKSUM` indicates that the NSR will be passing packets to the ND that require checksum insertion. If the NSR or upper level protocols will always generate all checksums for transmitted packets then there will be no checksum tags on transmit buffers and the ND does not need to check for those tags. By exchanging this information at

bind time, resource allocation and device initialization can be appropriately adjusted to account for checksum generation requirements

max_perfect_multicast is the maximum number of perfect multicast address matches that the Network Driver and associated hardware support. A perfect multicast address match is one which matches one and only one address (i.e. not a hashed match). This informs the NSR that there may be performance degradation if the ND is asked to match more multicast addresses than this perfect match limit.

If the NSR assigns more than one unicast address to the ND, then all unicast addresses beyond the first are also counted against the ***max_perfect_multicast*** limit to again avoid overloading the NIC hardware capabilities. In this case, `#_perfect_multicast + #_unicast - 1` must be no greater than ***max_perfect_multicast***.

max_total_multicast is the maximum number of multicast addresses that can be matched by the Network Driver and associated hardware. This is used to allow the NSR to determine how to handle situations where more multicast addresses are desired beyond hardware capabilities. The NSR must not, in any situation, issue a NIC control request that would set the total number of multicast addresses to be matched above this number.

mac_addr_len is the number of bytes to be used by the NSR for MAC addresses for this ND and overrides any default size for this media type. If specified as 0, the default MAC Address size is used based on the specified ***media_type***.

mac_addr is the current (factory) MAC address of the adapter being managed by the ND. The current MAC address of the adapter will be written to the first ***mac_addr_len*** bytes (or the number of bytes appropriate to that ***media_type*** if ***mac_addr_len*** is zero). The MAC address must be specified as an array of 8-bit binary values; each byte must be in host bit ordering format and not wire format; the bytes in the array must be in wire-format order.

DESCRIPTION

The `udi_nic_bind_cb_t` structure is used for the `udi_nsr_bind_req` and `udi_nsr_bind_ack` response operation between the parent (ND) and the child (NSR). This structure is “empty” on the `udi_nsr_bind_req` and the ND fills in the appropriate values to establish the parameters for the associated NIC Adapter for subsequent network data handling.

This control block must be declared by specifying the control block index value `UDI_NIC_BIND_CB_NUM` in a `udi_cb_init_t` in the driver’s `udi_init_info`.

The ***min_pdu_size*** and ***max_pdu_size*** arguments must not include space for any additional headers or trailers added by the ND.

REFERENCES

`udi_nsr_bind_ack`, `udi_init_info`, `udi_cb_init_t`, `udi_cb_alloc`

NAME **udi_nd_bind_req** *Network driver bind request*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nd_bind_req (
     udi_nic_bind_cb_t *cb,
     udi_index_t tx_chan_index,
     udi_index_t rx_chan_index);
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage bind control block. The fields of this control block are to be initialized by the ND and returned in the udi_nd_bind_ack request to specify the operational parameters for the ND.

tx_chan_index is the index of the data transmit channel which is to be created between the ND and the NSR by synchronizing via udi_channel_spawn operations.

rx_chan_index is the index of the data receive channel which is to be created between the ND and the NSR by synchronizing via udi_channel_spawn operations.

TARGET CHANNEL

The udi_nd_bind_req is issued to the ND over the Network Interface Metalanguage control channel which is the same as the primary channel created by the UDI Management Agent to initially bind the NSR and ND regions.

DESCRIPTION

udi_nd_bind_req is used to associate, or "bind" a channel between the NSR and the ND. The udi_nic_bind_req provides information for creating the data transfer channel(s) and any other configuration information for supporting this network adapter.

The udi_nd_bind_req must be the first operation sent to the ND by the NSR over a newly created bind channel. The NSR must then await the udi_nsr_bind_ack operation before issuing other operations.

The udi_nd_bind_req cannot be aborted.

This operation causes the ND and NSR to enter the BINDING state (see Section 2.4.1, "NIC Metalanguage States," on page 2-8).

REFERENCES

udi_nsr_bind_ack, udi_nic_bind_cb_t

NAME

udi_nsr_bind_ack *Network bind acknowledgment*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nsr_bind_ack (
     udi_nic_bind_cb_t *cb,
     udi_status_t status );
```

ARGUMENTS

cb is a pointer to the Network Driver bind acknowledgement control block.

status indicates the success or failure of the network bind request

TARGET CHANNEL

The `udi_nsr_bind_ack` is issued to the NSR via the Network Interface Metalanguage bind channel over which the `udi_nd_bind_req` operation was received.

DESCRIPTION

Handshakes `udi_nd_bind_req` operation to the NSR. The ND uses it to inform the NSR about the driver operational parameters and signify its readiness for network data transfer operations.

It is expected that the data transfer channels will have been fully spawned and configured before completion of the `udi_nsr_bind_ack` operation. The NSR will always issue the `udi_channel_spawn` for its end of the data transfer channels before issuing the `udi_nd_bind_req` operation.

The ND must use the same `udi_nic_bind_cb_t` obtained via the `udi_nd_bind_req` operation to generate the `udi_nsr_bind_ack` acknowledgement.

This operation causes the ND and NSR to enter the BOUND state (see Section 2.4.1, “NIC Metalanguage States,” on page 2-8).

STATUS VALUES

`UDI_OK` indicates that the network bind operation succeeded.

`UDI_STAT_INVALID_STATE` indicates that the ND is already bound to another NSR and cannot satisfy this bind request.

`UDI_STAT_HW_PROBLEM` indicates that the driver has detected a fatal hardware problem with the enumerated device and cannot complete the bind operation successfully.

REFERENCES

`udi_nd_bind_req, udi_nic_bind_cb_t`

NAME **udi_nd_unbind_req** *Network unbind request*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nd_unbind_req (
     udi_nic_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage standard control block.

TARGET CHANNEL

The udi_nd_unbind_req is issued to the ND over the Network Interface Metalanguage control channel.

DESCRIPTION

The udi_nd_unbind_req is used when the NSR wishes to close down an active binding between itself and the ND. All data transfer ceases, any queued data is discarded, and the data channel(s) are closed. The control channel may be used to issue a subsequent udi_nic_bind_req to the ND if desired or it may also be closed.

The udi_nd_unbind_req cannot be aborted.

This operation causes the ND and NSR to enter the UNBINDING state (see Section 2.4.1, “NIC Metalanguage States,” on page 2-8).

REFERENCES

udi_nsr_unbind_ack, udi_nic_cb_t

NAME

udi_nsr_unbind_ack *Network unbind acknowledgment*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nsr_unbind_ack (
     udi_nic_cb_t *cb,
     udi_status_t status );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage standard control block.

TARGET CHANNEL

The udi_nsr_unbind_ack is issued to the NSR over the Network Interface Metalanguage control channel. If the control block used for this operation is the one received in the corresponding udi_nd_unbind_req then the ***channel*** parameter will already be set to the correct value; this is the recommended mode of operation.

DESCRIPTION

Handshakes the network unbind request operation initiated by the NSR. As a result of this operation the routing information will be removed from the NSR and the control and data handling channels will be closed by both the ND and the NSR. No more data transfer will occur for this adapter instance unless another udi_nd_bind_req is issued by the NSR to create and bind a new set of data channels.

The udi_nsr_unbind_ack must use the same control block received with the udi_nd_unbind_req.

This operation causes the ND and NSR to enter the UNBOUND state (see Section 2.4.1, “NIC Metalanguage States,” on page 2-8).

STATUS VALUES

UDI_OK indicates that the network unbind request operation succeeded.

UDI_STAT_INVALID_STATE indicates that the ND has not completed a metalanguage bind with the NSR as setup by the udi_nd_bind_req and udi_nsr_bind_ack operations.

REFERENCES

udi_nd_unbind_req, udi_nic_cb_t

3.3.5 Interface Control

The interface control operations are used by the NSR to instruct the ND to enable or disable the network interface for link activity. Once the NSR and the ND have successfully been bound together, the `udi_nd_enable_req` operation is performed to activate the interface, usually as the result of a control operation by the system or user (e.g. ifconfig ... up).

The `udi_nd_enable_req` is successfully acknowledged by the ND using the `udi_nsr_enable_ack` operation as soon as the link activation operation is initiated. The acknowledgement is not expected to wait for the link to actually become enabled and the returned status in the `udi_nsr_enable_ack` is only used to indicated the success or failure of the ND in initiating the enable operation.

When the interface actually reports active, it will use the `udi_nsr_status_ind` operation to indicate that the link state has changed to the active state. Note that the interface may be enabled, but not necessarily have an active link status. Whenever the interface is in the enabled state, `udi_nsr_status_ind` operations must be used to indicate the link status if the link status changes.

The `udi_nd_disable_req` is used to disable an interface. The link status change may or may not be indicated by the `udi_nsr_status_ind` indication, but this request must perform the appropriate activity to force the interface into the disabled state. There is no acknowledgement to the NSR for the disable operation; the link must be disabled if currently active and there are no reasonable failure conditions.

When an interface is in the disabled state, link status changes must not be indicated by the `udi_nsr_status_ind` operation; if generated in this state, these indications will be ignored.

NAME

udi_nd_enable_req *Network link enable request*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nd_enable_req (
     udi_nic_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage standard control block.

TARGET CHANNEL

This request is issued to the ND over the Network Interface Metalanguage control channel. If the control block used for this operation has been received from a previous acknowledgement operation from the ND on this channel then the ***channel*** parameter in the control block already specifies the correct value; this is the recommended mode of operation.

DESCRIPTION

The `udi_nd_enable_req` is used when the NSR wishes to begin data transfer over the network connection provided by the ND and it's associated adapter. The NSR uses this operation to enable the link. The link will not necessarily become immediately available, although the ND will acknowledge this request as soon as it has successfully initiated the link enable operation. The actual link up state will be indicated by the ND via the `udi_nsr_status_ind` operation.

The ND is expected to update the configuration of the driver and adapter hardware from its device instance attributes (if any) each time the `udi_nd_enable_req` is issued. This insures that any device-specific configuration changes made by the MA or system administrator will take effect at that time.

This request cannot be aborted.

REFERENCES

`udi_nsr_enable_ack, udi_nd_disable_req, udi_nsr_status_ind, udi_nic_cb_t`

NAME

udi_nsr_enable_ack *Network link enable acknowledgment*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nsr_enable_ack (
     udi_nic_cb_t *cb,
     udi_status_t status );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage standard control block.

status indicates the success or failure of the enable operation

TARGET CHANNEL

This request is issued to the NSR over the Network Interface Metalanguage control channel. If the control block used for this operation is the one received in the corresponding `udi_nd_enable_req` then the ***channel*** parameter will already be set to the correct value; this is the recommended mode of operation.

DESCRIPTION

Handshakes network link enable request operation to the NSR. If successful, the NSR will subsequently expect to receive a `udi_nsr_status_ind` which indicates the actual link-up state for this network adapter connection.

Once the link-up state has been reached, the ND is expected to "post" transmit control blocks to the NSR for use in transmit requests by pre-allocating one or more of these control blocks and passing them to the NSR over the transmit channel using the `udi_nsr_tx_rdy` operation. The number of these transmit control blocks that are provided to the NSR represents the flow control level of the connection between the NSR and the ND; the NSR will never internally allocate transmit control blocks to be passed to the ND and will instead wait for the ND to supply these control blocks with the `udi_nsr_tx_rdy` operation.

This operation causes the ND and NSR to enter the ENABLED state (see Section 2.4.1, "NIC Metalanguage States," on page 2-8).

STATUS VALUES

`UDI_OK` indicates that the network enable request operation succeeded.

`UDI_STAT_HW_PROBLEM` indicates that the ND will not be able to enable the link due to a hardware problem.

`UDI_STAT_INVALID_STATE` indicates that the ND was not in the proper state and could not enable the hardware. This error code usually indictes a problem between the NSR and the ND.

WARNINGS

This operation must use the same control block as originally received from the `udi_nd_enable_req` operation.

REFERENCES

`udi_nsr_unbind_req`

NAME

udi_nd_disable_req *Network link disable request operation*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nd_disable_req (
     udi_nic_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage standard control block.

TARGET CHANNEL

This request is issued to the ND over the Network Interface Metalanguage control channel.

DESCRIPTION

The `udi_nd_disable_req` is used when the NSR wishes to shutdown an active link interface and terminate any data transfer activity for that link. This does not cause an unbind operation, and all control and data transfer channels between the NSR and the ND are preserved, but the hardware link interface state is set to remove the adapter from the network. This request must be honored and implemented by the ND; there is no failure case and correspondingly there is no acknowledgement to the NSR of this operation and the ND is responsible for deallocating this control block after processing the disable request.

The NSR may be notified of the link down state as a result of the ND issuing a `udi_nsr_status_ind`, but this is not required and the ND is not expected to communicate any link state changes to the NSR when the interface state is down.

When this operation is received by the ND, the ND should remove any requests queued to (but not yet acted upon by) the hardware and discard all currently held requests via appropriate actions. Operations which cannot be de-queued from the hardware are allowed to complete and should be handled on completion.

This request cannot be aborted.

This operation causes the ND and NSR to enter the BOUND state (see Section 2.4.1, “NIC Metalanguage States,” on page 2-8).

REFERENCES

`udi_nd_enable_req, udi_nic_cb_t`

3.3.6 Control and Status Operations

The purpose of the control operation is to perform a specific network-related function on the driver (ND) or the controlled hardware. The most common functions that have been identified are:

- setting a multicast address
- setting adapter physical address
- setting a filtering mode (i.e. promiscuous mode)
- resetting the driver/card

All of the above functions have been implemented using a single pair of channel operations: `udi_nd_ctrl_req` and `udi_nsr_ctrl_ack`. A command field in the *`udi_nic_ctrl_cb_t`* control block informs the ND of the type of control function the caller wishes to execute. The normal flow for the control operation is illustrated in Figure 3-4.

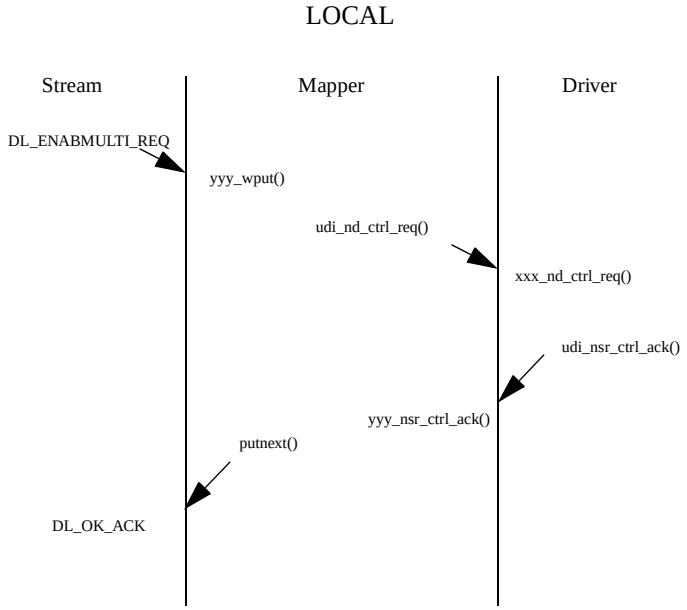

Figure 3-4 Control Operation (configure multicast address)

The information request/response operations are used by the NSR to obtain configuration and statistics information.

The status operation is an unsolicited event indication issued by the ND to the NSR to inform the NSR of a change in status. Some of the status events indicated by the `udi_nsr_status_ind` indications are similar to those that can be queried via the `udi_nd_ctrl_req` operation.

NAME

udi_nic_ctrl_cb_t *Network control operation control block*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

typedef struct {
     udi_cb_t gcb;
     udi_ubit8_t command;
     udi_ubit32_t indicator;
     udi_buf_t *data_buf;
} udi_nic_ctrl_cb_t;

/* Network Control Op Control Block Group Num */
#define UDI_NIC_CTRL_CB_NUM              3

/* Network Control Operation Commands */
#define UDI_NIC_ADD_MULTI                1
#define UDI_NIC_DEL_MULTI                2
#define UDI_NIC_ALLMULTI_ON              3
#define UDI_NIC_ALLMULTI_OFF             4
#define UDI_NIC_GET_CURR_MAC             5
#define UDI_NIC_SET_CURR_MAC             6
#define UDI_NIC_GET_FACT_MAC             7
#define UDI_NIC_PROMISC_ON               8
#define UDI_NIC_PROMISC_OFF              9
#define UDI_NIC_HW_RESET                 10
#define UDI_NIC_BAD_RXPKT                11
```

MEMBERS

gcb is the generic control block header which includes a pointer to the scratch space associated with this block and the channel context for the associated channel. The driver may use the scratch space while it owns the control block, but the values are not guaranteed to persist across channel operations.

command is the control command the driver must execute. The following control commands have been identified:

UDI_NIC_ADD_MULTI -- Configures the multicast addresses specified in the supplied buffer. The ***data_buf*** contains the array of MAC addresses to be added to the existing filter (the number of addresses being specified is indicated by the ***indicator*** field). The remaining portion of the ***data_buf*** contains the new filter table including the newly added addresses.

Devices which can modify their address filtering simply by knowing the addresses which are to be added or removed can use the initial ***indicator*** number of addresses to modify their filter. Other devices require access to the full table of addresses to be filtered to recompute the filter and may use

the latter portion of the ***data_buf***.

Although it is possible for the network stack or user-level applications to register multiple times for a specific multicast address, the NSR is responsible for handling this redundancy; the ND is only notified of multicast address modifications when the actual received packet filter must be modified. The NSR must also validate the multicast addresses to ensure that only valid multicast addresses are passed to the ND.

The contents of the data buffer are guaranteed to be preserved across a channel for the request but not for the response.

UDI_NIC_DEL_MULTI -- Removes the multicast addresses specified in the supplied buffer. In a manner similar to the UDI_NIC_ADD_MULTI operation, the initial ***indicator*** number of addresses in the ***data_buf*** lists the addresses to be removed, and the latter portion of the data_buf lists the new filter set after removing the specified addresses.

The contents of the data buffer are guaranteed to be preserved across a channel for the request but not for the response.

UDI_NIC_ALLMULTI_ON -- Enables the network adapter to receive all multicast addressed packets. The ***indicator*** field is unused and the ***data_buf*** field must be NULL. This operation additionally indicates that any specific multicast addresses registered (via UDI_NIC_ADD_MULTI) must be removed.

UDI_NIC_ALLMULTI_OFF -- Disables the reception of all multicast addresses. The ***indicator*** and ***data_buf*** arguments are used in the same manner as in the UDI_NIC_ADD_MULTI operation to specify the new list of specific multicast addresses (if any) that are to be passed to the NSR.

The adapter hardware is to be returned to the standard reception mode with no multicast addresses being received unless UDI_PROMISC_ON is in effect or unless specifically registered via this operation.

The contents of the data buffer are guaranteed to be preserved across a channel for the request but not for the response.

UDI_NIC_GET_CURR_MAC -- Reads the network adapter card's current physical MAC address or addresses. The array of MAC addresses is placed into the ***data_buf*** and the number of MAC addresses is placed into the ***indicator*** field in the same manner as described for the UDI_NIC_SET_CURR_MAC command. The ***buf_size*** in the ***data_buf*** divided by the ***indicator*** value should yield the same MAC address size as specified in the bind operation.

The contents of the data buffer are guaranteed to be preserved across a channel for the response but not for the request.

UDI_NIC_SET_CURR_MAC -- Configures the network adapter card's current physical MAC address or addresses. A typical card will have only a single unicast MAC address, but some configurations and hardware support the registration of multiple unicast MAC addresses for a network interface. The NSR may instruct the ND to register for multiple unicast MAC addresses by specifying an array of addresses in in the associated buffer. All unicast addresses beyond the first address will be accrued against the ND's ***max_perfect_multicast*** address count limit as specified in the bind operation; the total number of multicast and unicast addresses (excluding the first) registered by the NSR must not exceed the ***max_perfect_multicast*** value. The array of MAC addresses to be set is contained in the ***data_buf*** and the number of MAC addresses is in the ***indicator*** field. The ***buf_size*** in the ***data_buf*** divided by the ***indicator*** value should yield the same MAC address size as specified in the bind operation.

The contents of the data buffer are guaranteed to be preserved across a channel for the request but not for the response.

UDI_NIC_GET_FACT_MAC -- Reads the network adapter card's factory installed physical MAC address. The factory MAC address is specified as the single element array in the ***data_buf*** in the same manner as for the UDI_NIC_GET_CURR_MAC command. This is the initial MAC address used to operate the NIC until changed via UDI_NIC_SET_CURR_MAC.

The contents of the data buffer are guaranteed to be preserved across a channel for the response but not for the request.

UDI_NIC_PROMISC_ON -- Enables the promiscuous mode on the network adapter card. The ***indicator*** field is unused and the ***data_buf*** field must be NULL. When promiscuous mode is enabled, the hardware should be configured such that no destination address matching is performed and all packets should be received and sent to the NSR. Error packets will still be discarded by the ND unless UDI_NIC_BAD_RXPKT is used to specify otherwise.

UDI_NIC_PROMISC_OFF -- Disables the promiscuous mode on the network adapter card. The ***indicator*** field is unused and the ***data_buf*** field must be NULL. The NSR must now be passed only unicast addresses specific to this adapter or any multicast addresses enabled by any previous UDI_NIC_ADD_MULTI or UDI_NIC_ALLMULTI_ON operations.

UDI_NIC_HW_RESET -- Resets the network adapter card. The ***indicator*** field is unused and the ***data_buf*** field must be NULL. This operation should cause the hardware to be physically reset if possible. Any operations pending on the hardware should be cancelled and cleaned up; any operations pending in the driver that have not yet been delivered to the hardware (or which have already been completed by the hardware) should be processed as normal following the reset. All other activity should be suspended during the reset operation and the driver should restore the hardware to the same operational state that it had before the reset was issued with the following exceptions: promiscuous mode is disabled, no multicast addresses are registered, the current MAC address must be reprogrammed (i.e. the factory MAC address does not override the current setting), and link status should be established if the driver's state is ENABLED or ACTIVE when this request is received.

UDI_NIC_BAD_RXPKT -- Specifies ND handling of received packets which have an error indication. If the ***indicator*** is zero, the ND must simply discard the bad packet and await the next received packet.

If non-zero, the ***indicator*** specifies how many bytes of the bad packet should be passed to the NSR (with the appropriate rx_status indication). This is intended as a hint to allow the ND to terminate or discard reception of bad packets if needed; the bad packet is not required or guaranteed by the ND to match the length field in the ***indicator***. A receive indication with a bad packet status must be passed to the NSR even if no bytes from the received packet can be passed.

The ***data_buf*** field is unused and must be NULL.

indicator is a field which supplies additional information for the control operation as specified on a per-command basis.

data_buf is the data buffer associated with the control request, e.g., the buffer containing the multicast list addresses.

DESCRIPTION

The network control structure is used to configure, retrieve information or request a specific configuration action from the driver and/or the network adapter card. This structure is used with network control operations issued to the control channel.

Table 3-2 udi_nic_ctrl_cb_t Argument usage

Command (UDI_NIC_xxx)	Indicator	Data_buf contents
ADD_MULTI	number of new MAC addresses	MAC addresses (new/all)
DEL_MULTI	number of MAC addresses being removed	MAC addresses (removed/remaining)
ALLMULTI_ON	-	NULL
ALLMULTI_OFF	number of new MAC addresses	MAC addresses (new/all)
GET_CURR_MAC	returns number of MAC addresses	returns an array of MAC addresses
SET_CURR_MAC	number of MAC addresses	An array of MAC addresses
GET_FACT_MAC	returns number of MAC addresses	returns an array of MAC addresses
PROMISC_ON	-	NULL
PROMISC_OFF	-	NULL
HW_RESET	-	NULL
BAD_RXPKT	0 = ND discards bad RX packets non-zero = number of bytes of bad RX packets that the ND passes the NSR	NULL

The `UDI_NIC_PROMISC_ON` and `UDI_NIC_ALLMULTI_ON` shall be individually applied and the enabling or disabling of one shall not affect the setting of the other. The `UDI_NIC_PROMISC` settings do not affect the current multicast address table but the `UDI_NIC_ALLMULTI_ON` will delete the current list of explicit multicast addresses (which may be re-established in whole or in part by the `UDI_NIC_ALLMULTI_OFF`).

This control block must be declared by specifying the control block index value `UDI_NIC_CTRL_CB_NUM` in a `udi_cb_init_t` in the driver's `udi_init_info`.

The NSR obtains the `udi_nic_ctrl_cb_t` structure to use with the `udi_nd_ctrl_req` operation by calling `udi_cb_alloc` with a ***`cb_idx`*** that has been defined for the `UDI_NIC_CTRL_CB_NUM` control block.

REFERENCES

`udi_nd_ctrl_req`, `udi_init_info`, `udi_cb_init_t`, `udi_cb_alloc`

NAME

udi_nd_ctrl_req *Network control operation request*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nd_ctrl_req (
     udi_nic_ctrl_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage control operation control block.

TARGET CHANNEL

This request is issued to the ND driver over the Network Interface Metalanguage control channel. If the control block used for this operation has been received from a previous control acknowledgement operation from the ND on this channel then the ***channel*** parameter will already be set to the correct value; this is the recommended mode of operation.

DESCRIPTION

The `udi_nd_ctrl_req` operation is used to perform various network control functions on the ND and/or controlled adapter card.

The network control operations are described in more detail in the description of the `udi_nic_ctrl_cb_t` structure.

This request cannot be aborted.

REFERENCES

`udi_nsr_ctrl_ack, udi_nic_ctrl_cb_t`

NAME **udi_nsr_ctrl_ack** *Network control acknowledgment*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nsr_ctrl_ack (
     udi_nic_ctrl_cb_t *cb,
     udi_status_t status );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage control operation control block.

status indicates success or failure of the control request

TARGET CHANNEL This request is issued to the NSR over the Network Interface Metalanguage control channel. If the control block used for this operation has been received from a previous control request operation from the NSR on this channel then the ***channel*** parameter will already be set to the correct value; this is the recommended mode of operation.

DESCRIPTION The `udi_nsr_ctrl_ack` confirms the `udi_nd_ctrl_req` operation that has been issued to the driver and passes back any requested information in the control block or the associated buffer structure.

STATUS VALUES

`UDI_OK` indicates that the network control request operation succeeded.

`UDI_STAT_NOT_UNDERSTOOD` indicates that the control operation parameters were invalid.

`UDI_STAT_RESOURCE_UNAVAIL` indicates that the ND did not have and could not obtain the necessary resources to satisfy this request.

`UDI_STAT_HW_PROBLEM` indicates that the ND could not satisfy this request due to hardware problems or limitations.

WARNINGS This operation must use the same control block as originally received from the `udi_nd_ctrl_req` operation.

REFERENCES `udi_nd_ctrl_req, udi_nic_ctrl_cb_t`

NAME

udi_nic_status_cb_t *Status indication control block*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

typedef struct {
     udi_cb_t gcb;
     udi_ubit8_t event;
} udi_nic_status_cb_t;

/* Network Status Control Block Group Number */
#define UDI_NIC_STATUS_CB_NUM              4

/* Network Status Event Codes */
#define UDI_NIC_LINK_DOWN                  0
#define UDI_NIC_LINK_UP                    1
#define UDI_NIC_LINK_RESET                 2
```

MEMBERS

gcb is the generic control block header which includes a pointer to the scratch space associated with this block and the channel context for the associated channel. The driver may use the scratch space while it owns the control block, but the values are not guaranteed to persist across channel operations.

event is the status event code being indicated by the ND driver to the NSR. The following status events have been defined:

UDI_NIC_LINK_UP -- Link active transition. This event causes the ND and NSR to transition to the ACTIVE state (see Section 2.4.1, “NIC Metalanguage States,” on page 2-8).

UDI_NIC_LINK_DOWN -- Link inactive transition. This event causes the ND and NSR to transition to the ENABLED state (see Section 2.4.1, “NIC Metalanguage States,” on page 2-8).

UDI_NIC_LINK_RESET -- Link reset occurred. A link reset has the same effect as a UDI_NIC_LINK_DOWN event with the additional indication that any link state was lost and that a link-level error recovery was initiated. The NSR should be prepared to re-establish the link and remote node configuration information when a link reset occurs. This event also causes the ND and NSR to transition to the ENABLED state.

DESCRIPTION

The network status indication control block structure is used to notify the NSR of asynchronous events. This structure is used with network status indications issued across the control channel.

This control block must be declared by specifying the control block index value `UDI_NIC_STATUS_CB_NUM` in a `udi_cb_init_t` in the driver’s `udi_init_info`.

The ND obtains the `udi_nic_status_cb_t` structure to use with the `udi_nsr_status_ind` operation by calling `udi_cb_alloc` for the UDI_NIC_CONTROL_CB_NUM's declared ***`cb_idx`*** value.

REFERENCES

`udi_nsr_status_ind, udi_init_info, udi_cb_init_t, udi_cb_alloc`

NAME

udi_nsr_status_ind *Network status indication*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nsr_status_ind (
     udi_nic_status_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage status indication control block.

TARGET CHANNEL

This request is issued to the NSR over the Network Interface Metalanguage control channel.

DESCRIPTION

The `udi_nsr_status_ind` indication is used by the ND to asynchronously notify the NSR of various status information such as any change in link status. The NSR should evaluate the status indication, perform the appropriate action, and then deallocate the status indication control block.

The ND is responsible for allocating the `udi_nic_status_cb_t` for this operation since this operation originates in the ND and is not a response to an NSR request. The ND may not abort this operation once it is issued to the NSR.

REFERENCES

`udi_nic_status_cb_t`

NAME **udi_nic_info_cb_t** *Network information control block*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

typedef struct {
     udi_cb_t gcb;
     udi_boolean_t interface_is_active;
     udi_boolean_t link_is_active;
     udi_boolean_t is_full_duplex;
     udi_ubit32_t link_mbps;
     udi_ubit32_t link_bps;
     udi_ubit32_t tx_packets;
     udi_ubit32_t rx_packets;
     udi_ubit32_t tx_errors;
     udi_ubit32_t rx_errors;
     udi_ubit32_t tx_discards;
     udi_ubit32_t rx_discards;
     udi_ubit32_t tx_underrun;
     udi_ubit32_t rx_overrun;
     udi_ubit32_t collisions;
} udi_nic_info_cb_t;

/* Network Information Control Block Group Number */
#define UDI_NIC_INFO_CB_NUM              5
```

MEMBERS

gcb is the generic control block header which includes a pointer to the scratch space associated with this block and the channel context for the associated channel. The driver may use the scratch space while it owns the control block, but the values are not guaranteed to persist across channel operations.

interface_is_active is a boolean indication of whether the interface has been activated or not (via `udi_nd_enable_req`).

link_is_active is a boolean indication of whether the link is active or not (line status).

is_full_duplex is the boolean indication of whether the link is operating in full-duplex or half-duplex mode. Note that the `%duplex` custom attribute may be used to indicate the desired duplex mode but this field will always report the current duplex mode (see Table 2-4, “NIC Custom Attributes,” on page 2-4).

link_mbps is the current link data rate in megabits-per-second. Note that the `%speed_mbps` custom attribute may be used to indicate the desired speed but this field will always report the current data link speed (see Table 2-4, “NIC Custom Attributes,” on page 2-4).

link_bps is the current link data rate in bits-per-second. This field is used in the same manner that the ***link_mbps*** field is used but specifies a lower range of values.

tx_packets is the number of packets transmitted by this interface (inclusive of discarded packets and those in which the transmit request completed with errors).

rx_packets is the number of packets received by this interface (inclusive of discarded packets and those indicating receive errors).

tx_errors is the number of packet transmissions that encountered an error of some form.

rx_errors is the number of receive packet operations that encountered an error of some form.

tx_discards is the number of packets which were discarded before transmission for internal reasons (e.g. timeouts or unavailable resources).

rx_discards is the number of packets which were discarded after being received but before being passed to the NSR for internal reasons (e.g. unavailable resources).

tx_underrun is the number of transmit underrun errors (which is also included in ***tx_errors***).

rx_overrun is the number of receive overrun errors (which is also included in ***rx_errors***).

collisions is the number of transmit packet collisions (if applicable).

DESCRIPTION

The network information control block structure is used to provide configuration and statistics information to the NSR.

Fields which represent counters are subject to overflow and will silently wraparound to continue counting from zero. The statistics values should be read frequently enough to detect this wraparound condition if it is significant.

The ***link_bps*** field is customarily used for slower WAN protocols where the link data rate may range from 300 bps to 128 Kbps. The ***link_mbps*** field is used for higher-speed LAN protocols. Note that there is an overlap between these two fields; it is expected that if the ***link_mpbs*** field is non-zero that the ***link_bps*** field may be ignored. These values are supplied to provide diagnostic information and allow predictive scheduling of data exchange and therefore are not required to be exact nor do they control the associated hardware settings.

This control block must be declared by specifying the control block index value `UDI_NIC_INFO_CB_NUM` in a `udi_cb_init_t` in the driver's `udi_init_info`.

The NSR or ND obtains the `udi_nic_bind_req_cb_t` structure to use with the `udi_nd_info_req` and `udi_nsr_info_ack` operations by calling `udi_cb_alloc` with a ***`cb_idx`*** that has been defined for the `UDI_NIC_INFO_CB_NUM` control block.

REFERENCES

`udi_nd_info_req, udi_nsr_info_ack, udi_init_info, udi_cb_init_t, udi_cb_alloc`

NAME

udi_nd_info_req *Network information request*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nd_info_req (
     udi_nic_info_cb_t *cb,
     udi_boolean_t reset_statistics );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage information control block.

reset_statistics is a boolean value which indicates whether the ND is to reset the statistics counters or not.

TARGET CHANNEL

This request is issued to the ND over the Network Interface Metalanguage control channel. If the control block used for this operation has been received from a previous acknowledgement operation from the ND on this channel then the ***channel*** parameter will already be set to the correct value; this is the recommended mode of operation.

DESCRIPTION

The `udi_nd_info_req` is used by the NSR to request information from the NIC Driver. The NSR supplies a `udi_nic_info_cb_t` control block which is filled in with the appropriate information by the ND and then passed back to the NSR.

The metalanguage-specific fields in the control block are unused for `udi_nd_info_req` and are intended to be filled in by the ND for `udi_nsr_info_ack`.

If the ***reset_statistics*** argument is true, the ND should reset the statistics after providing their current values in the ***cb*** control block being returned.

This request cannot be aborted.

REFERENCES

`udi_nic_info_cb_t, udi_nsr_info_ack`

NAME **udi_nsr_info_ack** *Network information response*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nsr_info_ack (
     udi_nic_info_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage information control block.

TARGET CHANNEL This request is issued to the NSR over the Network Interface Metalanguage control channel.

DESCRIPTION The `udi_nsr_info_ack` is the ND response to the `udi_nd_info_req` issued by the NSR. The ND passes back the information in the fields of the associated control block.

WARNINGS This operation must use the same control block as originally received from the `udi_nd_info_req` operation.

REFERENCES `udi_nic_info_cb_t, udi_nd_info_req`

3.4 Data Transfer Operations

These operations are used to allow the local protocol stack to communicate with other hosts on the network by passing requests to or from the ND via the data transfer channels connected to the NSR. The associated data buffers for these requests are the data “packets” to be sent over the network and are assumed (by the ND) to have the proper network headers already constructed and provided, although the ND is free to add device or network-specific headers as needed.

If the semantics of the underlying network technology are based on unacknowledged broadcast methodologies (e.g. Ethernet) then the ND will acknowledge transmit requests back to the NSR as soon as the ND has completed the transmit operation itself. If the network technology incorporates packet or frame level acknowledgement[4], this should be managed by the ND and the transmit operations should not be acknowledged to the NSR until the remote host has acknowledged them back to the ND. The ND is not expected to implement retransmissions in either case unless desirable by the ND implementation or the network technology.

The ND is also expected to generate packet or frame level acknowledgements if so required by the network technology.

Any technology-specific operations not performed by the hardware must be implemented by the ND internally (e.g. ATM protocol segmentation/reassembly of packets into ATM cells).

The normal sequence of events for data transfer is illustrated in Figure 3-5.

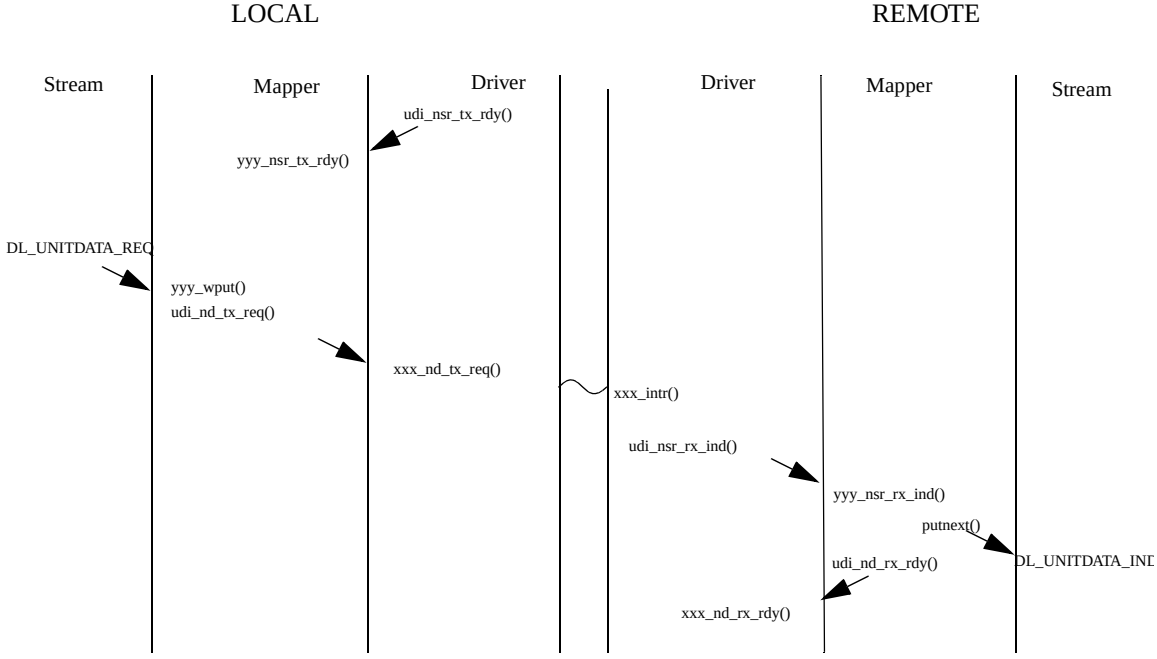

Figure 3-5 Connectionless data transfer operation

4. Packet or frame level acknowledgement is differentiated from protocol-level acknowledgement; the latter is implemented by higher layers of the protocol stack than that represented by the ND. An example of protocol-level acknowledgement is the TCP ACK packet.

NAME **udi_nic_tx_cb_t** *Network transmit control block*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

typedef struct {
     udi_cb_t gcb;
     udi_nic_tx_cb_t *chain;
     udi_buf_t *tx_buf;
     udi_boolean_t completion_urgent;
} udi_nic_tx_cb_t;

/* Network Transmit Control Block Group Number */
#define UDI_NIC_TX_CB_NUM                6
```

MEMBERS

gcb is the generic control block header which includes a pointer to the scratch space associated with this block and the channel context for the associated channel. The driver may use the scratch space while it owns the control block, but the values are not guaranteed to persist across channel operations.

chain is a pointer to the next `udi_nic_tx_cb_t` structure (and associated packet buffer) for this operation. The ND and NSR will use this field to "batch" a number of transmit requests or ready's into a single metalanguage operation. The ND, NSR, or environment are free to divide the chain at any point and implement explicit operations for each resulting portion of the chain, but performance concerns would indicate that processing the entire chain as a batch is highly desirable. The end of the chain is indicated by a NULL pointer.

tx_buf is the buffer describing the packet to be transmitted This field must be NULL for the `udi_nsr_tx_rdy` operation and is used only for the `udi_nd_tx_req` and `udi_nd_exp_tx_req` operations, for which the contents of the buffer are guaranteed to be preserved across a channel.

completion_urgent is a hint to the ND that the NSR or UDI environment considers the associated packet buffer to be a critical resource and that it should be returned (via `udi_buf_free`) as quickly as possible after the transmit completes. This field is ignored for the `udi_nsr_tx_rdy` operation.

DESCRIPTION

The `udi_nic_tx_cb_t` structure is passed from the ND to the NSR to indicate a capability for transmitting a packet; the NSR subsequently attaches a packet buffer to this control block and returns it to the ND for transmission. The NSR can only pass packets to the ND when it has available control blocks, thereby implementing flow control between the ND and the NSR by requiring the ND to supply the NSR with all usable `udi_nic_tx_cb_t` structures (*i.e.* the NSR must *not* allocate control blocks of this type).

This control block must be declared by specifying the control block index value `UDI_NIC_TX_CB_NUM` in a `udi_cb_init_t` in the driver's `udi_init_info`.

The ND obtains the `udi_nic_tx_cb_t` structure by calling `udi_cb_alloc` with a ***`cb_idx`*** that has been defined for the `UDI_NIC_TX_CB_NUM` control block.

REFERENCES

`udi_nsr_tx_rdy`, `udi_nd_tx_req`, `udi_init_info`, `udi_cb_init_t`, `udi_cb_alloc`

NAME

udi_nsr_tx_rdy *Network driver ready to transmit packet*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nsr_tx_rdy (
     udi_nic_tx_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage transmit control block.

TARGET CHANNEL

This request is issued to the NSR over the Network Interface Metalanguage transmit data channel.

DESCRIPTION

The `udi_nsr_tx_rdy` is used to indicate to the NSR that the ND can transmit a packet when the NSR has a packet available. When the NSR receives a packet from the protocol stack that should be transmitted, it looks for a `udi_nic_tx_cb_t` to associate with that packet. If no `udi_nic_tx_cb_t` is available, it must queue the packet and exert flow control "back pressure" into the protocol stack. When a `udi_nic_tx_cb_t` is available (either immediately or as a result of this command), the NSR may use that structure to pass the packet buffer to the ND for transmission.

The ND may chain multiple `udi_nic_tx_cb_t` structures together as a linked list via the ***`chain`*** field in the structure and pass the entire chain to the NSR in a single `udi_nsr_tx_rdy` operation.

Following a `udi_nd_disable_req`, the NSR should return all transmit control blocks to the ND driver. This is done via the `udi_nd_tx_req` transmit path, but there will be no associated buffer for these operations.

Following a `udi_nd_unbind_req` or a `udi_channel_closed` operation, the ND and NSR are each responsible for deallocating the transmit control blocks they are holding; no transmit control blocks are passed between the ND and NSR following these operations.

The ND may increase or decrease the number of transmit opportunities available to the NSR at any point in time by making correspondingly more or fewer transmit control blocks available to the NSR. The NSR should not expect the number of transmit control blocks to stay constant over the lifetime of the ND instance.

This request cannot be aborted.

REFERENCES

`udi_nd_tx_req, udi_nic_tx_cb_t`

NAME

udi_nd_tx_req *Network send packet*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nd_tx_req (
     udi_nic_tx_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage transmit control block.

TARGET CHANNEL

This request is issued to the ND over the Network Interface Metalanguage transmit data channel.

DESCRIPTION

The `udi_nd_tx_req` is called by the NSR to pass one or more packet buffers to the ND for transmission on the network. The NSR has already build the datalink frame header for the packet and obtained a `udi_nic_tx_cb_t` from the ND to associate with the packet. The NSR issues the `udi_nd_tx_req` to the ND; once the packet has been successfully transmitted the ND will deallocate the packet buffer and pass the `udi_nd_tx_cb_t` back to the NSR in the `udi_nsr_tx_rdy` operation.

The NSR may chain multiple `udi_nic_tx_cb_t` structures together as a linked list via the `chain` field in the structure and pass the entire chain to the ND in a single `udi_nd_tx_req` operation; each `udi_nic_tx_cb_t` has an associated buffer which represents a separate packet. When passing the `udi_nic_tx_cb_t` structures back to the NSR via the `udi_nsr_tx_rdy` operation the ND is free to subdivide the chain and pass it back in pieces, or it may return the entire chain at once.

Each `udi_nd_tx_cb_t` contains a completion urgency hint field which is used to indicate to the ND that the corresponding packet transmission should be completed as quickly as possible to return the packet buffer. If this hint does not indicate any packet transmission urgency, the ND is not required to detect transmission completion in any set time period.

The ND is responsible for removing the buffer from the `net_tx_cb` structure and deallocating the buffer (`udi_buf_free`) before posting the structure back to the NSR via the `udi_nsr_tx_rdy` operation.

Note that in the special case where the NSR has initiated a `udi_nsr_disable_req` operation to this ND, it will return any transmit control blocks it owns via the `udi_nd_tx_req` operations; these control blocks have no attached buffers and are simply being recycled to the ND device.

REFERENCES

`udi_nsr_tx_rdy, udi_nd_exp_tx_req`

NAME **udi_nd_exp_tx_req** *Expedited data transmit request*

SYNOPSIS

```
#include <udi.h>

void udi_nd_exp_tx_req (
     udi_nic_tx_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage transmit control block.

TARGET CHANNEL

This request is issued to the ND over the Network Interface Metalanguage transmit data channel.

DESCRIPTION

The `udi_nd_exp_tx_req` is called by the NSR to pass one or more packet buffers to the ND for expedited transmission on the network. This operation is functionally equivalent to the `udi_nd_tx_req` except that the packet(s) associated with this request are "high-priority" and should be handled immediately by the ND, before any current or future low-priority traffic is handled.

REFERENCES

`udi_nd_tx_req, udi_nsr_tx_rdy, udi_nic_tx_cb_t`

NAME

udi_nic_rx_cb_t *Network receive control block*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

typedef struct {
    udi_cb_t gcb;
    udi_nic_rx_cb_t *chain;
    udi_buf_t *rx_buf;
    udi_ubit8_t rx_status;
    udi_ubit8_t addr_match;
    udi_ubit8_t rx_valid;
} udi_nic_rx_cb_t;

/* Network Receive Control Block Group Number */
#define UDI_NIC_RX_CB_NUM                7

/* values for rx_status */
#define UDI_NIC_RX_BADCKSUM              (1U<<0)
#define UDI_NIC_RX_UNDERRUN              (1U<<1)
#define UDI_NIC_RX_OVERRUN               (1U<<2)
#define UDI_NIC_RX_DRIBBLE               (1U<<3)
#define UDI_NIC_RX_FRAME_ERR             (1U<<4)
#define UDI_NIC_RX_MAC_ERR               (1U<<5)
#define UDI_NIC_RX_OTHER_ERR             (1U<<7)

/* values for addr_match */
#define UDI_NIC_RX_UNKNOWN               0
#define UDI_NIC_RX_EXACT                 1
#define UDI_NIC_RX_HASH                  2
#define UDI_NIC_RX_BROADCAST             3

/* values for rx_valid */
#define UDI_NIC_RX_GOOD_IP_CKSUM         (1U<<0)
#define UDI_NIC_RX_GOOD_TCP_CKSUM        (1U<<1)
#define UDI_NIC_RX_GOOD_UDP_CKSUM        (1U<<2)
```

MEMBERS

gcb is the generic control block header which includes a pointer to the scratch space associated with this block and the channel context for the associated channel. The driver may use the scratch space while it owns the control block, but the values are not guaranteed to persist across channel operations.

chain is a pointer to the next `udi_nic_rx_cb_t` structure (and associated packet buffer) for this operation. The ND and NSR will use this field to "batch" a number of identical receive indications or responses into a single metalanguage operation. The ND, NSR, or environment are free to divide the chain at any point and implement explicit operations for each resulting

portion of the chain, but performance concerns would indicate that processing the entire chain as a batch is highly desirable. The end of the chain is indicated by a NULL pointer.

rx_buf is the buffer containing the packet that has been received. This field is used to pass an empty buffer (***rx_buf***->buf_size = 0) to the ND in the udi_nd_rx_rdy operation which will be filled in with an incoming packet and returned (buffer data is preserved) with the udi_nsr_rx_ind or udi_nsr_exp_rx_ind indications.

rx_status is a bitmask indicating if there were any problems with the received packet. This field supports the following bit values:

UDI_NIC_RX_BADCKSUM - Indicates that the adapter hardware or associated driver performed checksum calculation and validation and determined that the calculated checksum value for the packet did not match one or more of the corresponding checksum values indicated internally in the packet. If this status is not indicated for the packet it only indicates that there was no specific checksum failure detected by the ND or the adapter but not that the checksums have been validated to be correct.

UDI_NIC_RX_UNDERRUN - Indicates that the packet is too short (did not meet minimum PDU size requirements) or that the reception terminated before the entire packet was received. Also known as a runt packet.

UDI_NIC_RX_OVERRUN - Indicates that the packet is too large (exceeds maximum PDU size requirements).

UDI_NIC_RX_DRIBBLE - Indicates that the frame contained a non-integer multiple of 8 bits (“dribbling” bits). The disposition of the dribbled bits is indeterminate: the hardware may or may not provide them to the ND, therefore the NSR should not examine the data for the dribble value.

UDI_NIC_RX_FRAME_ERR - Indicates that the packet had a framing error. These are technology-specific but include such causes as: incorrect frame header characters, incorrect frame trailer characters, invalid characters in the frame header or trailer, or an invalid CRC value for the frame.

UDI_NIC_RX_MAC_ERR - Indicates that the packet was damaged by a media access error. These types of errors are network specific. A typical use of this bit is for a late collision that has occured after the valid collision period for a broadcast media technology (e.g. Ethernet).

UDI_NIC_RX_OTHER_ERR - Indicates that some type of error was detected by the adapter or driver that does not fit into one of the other error classes. The ND is not expected to signal errors related to the operation of the local hardware or the driver; the ***rx_status*** field should be used only for packet or network related errors encountered during receives.

When the ND indicates a receive packet error via a non-zero ***rx_status*** value, no guarantees are made regarding whether data will be returned in the associated buffer and any data present should only be used for diagnostic purposes; under no circumstances must this data be used as part of the normal transfer operations.

addr_match is an indicator of the ND's knowledge of the MAC address match for the current packet. The ND is not required to know how the receive packet matched the set of acceptable addresses, but if a determination can easily be made from hardware registers or additional packet information, the ND can supply this information to the NSR in this field to optimize processing of the packet.

Valid values for this field are:

UDI_NIC_RX_UNKNOWN - The ND does not know how the received packet matched

UDI_NIC_RX_EXACT - The received packet exactly matched a valid unicast or multicast address for which the ND and associated adapter were registered.

UDI_NIC_RX_HASH - The received packet matched based on a hashing algorithm and the address needs to be further verified by the NSR at some point.

UDI_NIC_RX_BROADCAST - The received packet used the media broadcast address.

The ***addr_match*** field is a hint to the NSR to optimize processing of the packet, therefore the ND must never indicate any type of match that is not known to be valid (e.g. indicate an EXACT match when it was a BROADCAST). When in doubt, the ND should always use the UNKNOWN indicator.

rx_valid indicates that one or more aspects of this receive have been successfully validated:

UDI_NIC_RX_GOOD_IP_CKSUM - indicates that the IP checksum was validated and found to be correct for the received packet.

UDI_NIC_RX_GOOD_TCP_CKSUM - indicates that the TCP checksum was validated and found to be correct for the received packet.

UDI_NIC_RX_GOOD_UDP_CKSUM - indicates that the UDP checksum was validated and found to be correct for the received packet.

If the checksum value is also known, it may be attached to the buffer via the UDI_BUFTAG_BE16_CHECKSUM tag; if the NSR has indicated at bind time that the checksum will not be used (by not setting UDI_NIC_CAP_USE_RX_CKSUM) then the ND should not bother to set the tag, even if the checksum value is known.

DESCRIPTION

The udi_nic_rx_cb_t structure is used to pass packets from the ND to the NSR after they have been received. They are provided to the ND via the udi_nd_rx_rdy operation.

These structures are supplied to the ND by the NSR to be attached to received packets. If the NSR has not supplied the ND with one of these control blocks to attach to a received data packet, the ND is flow-blocked and may not allocate one of these control blocks to handle the incoming data. It may save or preserve the data but it cannot pass more data to the NSR until the NSR has supplied available udi_nic_rx_cb_t structures.

This control block must be declared by specifying the control block index value UDI_NIC_RX_CB_NUM in a udi_cb_init_t in the driver's udi_init_info.

The NSR obtains the udi_nic_rx_cb_t structure by calling udi_cb_alloc with a ***cb_idx*** that has been defined for the UDI_NIC_RX_CB_NUM control block.

If the NSR uses the ***rx_valid*** field for checking the validity of the received packet, it must zero this field before passing the control block to the ND. The ND must not set these bits unless the checksum has been validated and the packet is of the appropriate type (e.g. the UDI_NIC_RX_GOOD_TCP_CKSUM bit must not be set if this is not a TCP packet). The use of this field allows performance optimizations when handling this packet; neither the ND nor the NSR is required to utilize this field.

REFERENCES

udi_nsr_rx_ind, udi_nsr_exp_rx_ind, udi_nd_rx_rdy, udi_init_info, udi_cb_init_t, udi_cb_alloc

NAME

udi_nsr_rx_ind *Network receive packet indication*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nsr_rx_ind (
     udi_nic_rx_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage receive control block.

TARGET CHANNEL

This request is issued to the NSR over the Network Interface Metalanguage receive data transfer channel.

DESCRIPTION

The `udi_nsr_rx_ind` is used when the ND has received a packet and is passing it to the NSR for handling by the protocol stack. The ***cb*** can be an individual receive control block or a chain of receive control blocks; each control block references a separate received packet. The packet buffer will be parsed and then demultiplexed to the interested protocol stack entities by the NSR, and the ***cb*** will subsequently be returned via the `udi_nd_rx_rdy` operation.

WARNINGS

The control block must be one that was passed to the ND previously via the `udi_nd_rx_rdy` operation.

REFERENCES

`udi_nd_rx_rdy, udi_nsr_exp_rx_ind`

NAME **udi_nsr_exp_rx_ind** *Network receive packet indication*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nsr_exp_rx_ind (
     udi_nic_rx_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage receive control block.

TARGET CHANNEL This request is issued to the NSR over the Network Interface Metalanguage receive data transfer channel.

DESCRIPTION The `udi_nsr_exp_rx_ind` is used when the ND has received an expedited (high-priority) packet and is passing it to the NSR for handling by the protocol stack. The functionality of this operation is identical to the `udi_nsr_rx_ind` operation except that this interface must be used if the packet was determined to be urgent or high-priority.

WARNINGS The control block must be one that was passed to the ND previously via the `udi_nd_rx_rdy` operation.

REFERENCES `udi_nd_rx_rdy, udi_nsr_rx_ind`

NAME

udi_nd_rx_rdy *Network receive packet response*

SYNOPSIS

```
#include <udi.h>
#include <udi_nic.h>

void udi_nd_rx_rdy (
     udi_nic_rx_cb_t *cb );
```

ARGUMENTS

cb is a pointer to a Network Interface Metalanguage receive control block.

TARGET CHANNEL

This request is issued to the ND over the Network Interface Metalanguage receive data transfer channel. If the NSR re-uses a control block previously returned to it by the ND then the ***channel*** parameter in that control block will already correctly identify the target channel.

DESCRIPTION

The udi_nd_rx_rdy is used by the NSR to provide one or more udi_nic_rx_cb_t structures and associated receive buffers to the ND. The ND may then use these buffers to receive packets from the network and pass those packets to the NSR in udi_nsr_rx_ind or udi_nsr_exp_rx_ind indications. The ND must not allocate udi_nic_rx_cb_t control blocks internally because these are regulated by the NSR to implement flow control.

The ND must also refrain from allocating receive buffers internally if possible. In any event, the incoming data must always be passed to the NSR in the same buffer (or a direct udi_buf_write/udi_buf_copy version thereof) that was originally passed to the ND by the NSR. If the ND has initially received the packet into a separate buffer it must issue a udi_buf_copy or udi_buf_write operation to copy that packet into the buffer provided by the NSR to send that packet to the NSR.

The NSR will not expect the ND to preserve or receive packets during any period of time wherein the NSR has not provided the ND with enough receive control blocks and buffers to accomodate the incoming data. It is recommended but not required that the NSR provide the ND with buffers whose length is zero and that it provide at least ***rx_hw_threshold*** number of receive control blocks at any point in time. The ND must set the buffer size to the needed size (usually the ***max_pdu_size*** from the udi_nic_bind_cb_t) before calling udi_buf_dma_map.

The ND should return all of the control blocks to the NSR with a zero-length buffer upon receipt of a udi_nd_disable_req operation.

Note that if a udi_nd_unbind_req or udi_channel_closed operation occurs, the NSR will not return the receive control blocks or buffers to the ND device and will deallocate them internally.

This operation cannot be aborted.

REFERENCES

udi_nsr_rx_ind, udi_nsr_exp_rx_ind

Index