[MS-RNDIS]:
Remote Network Driver Interface Specification (RNDIS) Protocol

---

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names.** The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
09/25/2009	0.1	Major	First Release.
11/06/2009	0.1.1	Editorial	Revised and edited the technical content.
12/18/2009	0.1.2	Editorial	Revised and edited the technical content.
01/29/2010	0.2	Minor	Updated the technical content.
03/12/2010	0.3	Minor	Updated the technical content.
04/23/2010	0.3.1	Editorial	Revised and edited the technical content.
06/04/2010	0.3.2	Editorial	Revised and edited the technical content.
07/16/2010	0.3.2	No change	No changes to the meaning, language, or formatting of the technical content.
08/27/2010	0.3.2	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2010	0.3.2	No change	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	0.3.2	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	0.3.2	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	0.3.2	No change	No changes to the meaning, language, or formatting of the technical content.
03/25/2011	0.3.2	No change	No changes to the meaning, language, or formatting of the technical content.
05/06/2011	0.3.2	No change	No changes to the meaning, language, or formatting of the technical content.
06/17/2011	0.4	Minor	Clarified the meaning of the technical content.
09/23/2011	0.4	No change	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	1.0	Major	Significantly changed the technical content.
03/30/2012	1.0	No change	No changes to the meaning, language, or formatting of the technical content.
07/12/2012	1.0	No change	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	1.0	No change	No changes to the meaning, language, or formatting of

Date	Revision History	Revision Class	Comments
			the technical content.
01/31/2013	1.0	No change	No changes to the meaning, language, or formatting of the technical content.
08/08/2013	2.0	Major	Significantly changed the technical content.
11/14/2013	3.0	Major	Significantly changed the technical content.
02/13/2014	4.0	Major	Significantly changed the technical content.
05/15/2014	5.0	Major	Significantly changed the technical content.

Contents

1 Introduction

This document specifies the Remote Network Driver Interface Specification (RNDIS) Protocol. The Remote Network Driver Interface Specification Protocol, referred to also as RNDIS in this document defines the communication between a host and **network device** connected over an external **bus transport** such as Universal Serial Bus (**USB**), so that the host can obtain network connectivity through the RNDIS-compliant device. The protocol enables the host to provide a vendor-independent **class driver** for an RNDIS compliant network device.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

bus
device
device driver
host(1)
little-endian
padding
resource
universal serial bus (USB)

The following terms are specific to this document:

bus transport: A **bus** mechanism that can transport bits between the host and the **device**, such as **USB**.

channel: An independent communication mechanism on the **bus transport** between the host and the connected **device**.

class driver: A vendor independent **device driver** running on the host operating system that supports any device that conforms to the device interface specified by the host operating system.

control channel: The communication **channel** on the **bus transport** between the host and the device on which **control messages** are sent.

control message: A message used to control the **device**.

data channel: The communication **channel** on the **bus transport** between the host and the **device** on which networking data is sent and received.

data message: A message containing the network packet data.

flow control: The mechanism to rate-limit the messages from the sender so as to not overflow the buffers on the receiver.

NDIS: Network Driver Interface Specification, which allows different networking components of an operating system to interoperate with one another, including the **device drivers** for the **networking devices** connected to the system.

network device: A **device** that can provide network access for a host.

OID: Object **Id**entifier: A 4-byte integer used to identify a specific property of the device, as specified by the host operating system.

out-of-band data: Any special data that can be sent independent of the network data, such as packet priority.

per-packet-info data: Any metadata associated with the packet such as the TCP Checksum and so on.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

1.2.2 Informative References

[IEEE802.3-2008] Institute of Electrical and Electronics Engineers, "Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Description", IEEE Std 802.3, 2008, http://standards.ieee.org/getieee802/802.3.html

[IEEE802.11-2007] Institute of Electrical and Electronics Engineers, "Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", ANSI/IEEE Std 802.11-2007, http://standards.ieee.org/getieee802/download/802.11-2007.pdf

Note There is a charge to download this document.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

[MSDN-NDIS] Microsoft Corporation, "NDIS Library Function References", http://msdn.microsoft.com/en-us/library/ff557045.aspx

[MSDN-OIDs] Microsoft Corporation, "Remote NDIS OIDs", http://msdn.microsoft.com/en-us/library/ff570633.aspx

[MSDN-RNDISUSB] Microsoft Corporation, "Remote NDIS To USB Mapping", http://msdn.microsoft.com/en-us/library/ff570657.aspx

[USB-SPC] USB Consortium, "USB 3.0 Specification", April 2000, http://www.usb.org/developers/docs/

1.3 Overview

Network **device drivers** are complex to implement and maintain, and are not always readily available in the **host** operating system. RNDIS attempts to alleviate these problems by eliminating the need for a vendor-supplied network device driver. Instead, the host leverages a **class driver** which supports any vendor's **device** that is conformant to the RNDIS Protocol.

RNDIS is a message-based protocol that can be implemented over any external bus transport such as USB. The **bus** should be capable of supporting separate control and **data channels** that are reliable and should offer sequential delivery of messages between the host and the device.

The RNDIS Protocol consists of a message set and a sequence of message exchanges between the host and the device to facilitate device configuration, querying device parameters, indicating device/network status change, and sending/receiving network data. The following diagram shows the general architecture of the RNDIS Protocol.

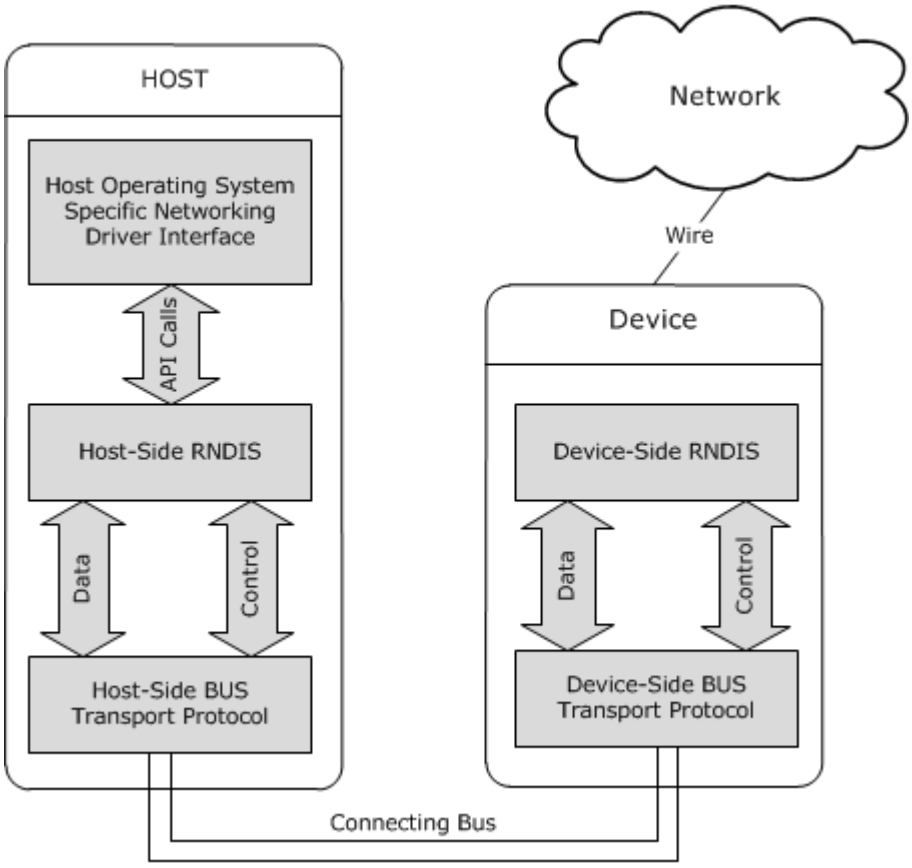


Figure 1: General architecture of the RNDIS Protocol

The host side of the protocol is responsible for the following:

- Initializing/deinitializing the device, as specified in section 3.1.3.
- Establishing the data and **control channels** between the host and the device over the bus transport, as specified in section 3.1.3.
- Exchanging of **data messages** and **control messages** between the host and the device as mandated by the host operating system's specific network driver interface, as specified in section 3.1.4.
- Encapsulating and transferring of the data packets over the bus transport, as mandated by the supported bus specification, as described in section 1.4.

The device side of the protocol is responsible for the following:

- Interpreting the control messages received on the bus as defined by the bus protocol and appropriately responding to them, as specified in sections 3.2.3 and 3.2.5. This includes handling requests for device initialization/deinitialization and establishing control/data channels.
- Indicating any network or device-related status to the host operating system's specific network driver interface, as specified in section 3.2.7.
- Exchanging of data messages between the host and the device, as specified in section 3.2.5.7.
- Exercising **flow control** as mandated by the underlying bus to prevent the host from overflowing the data buffers of the device, as described in section 1.4.

1.4 Relationship to Other Protocols

RNDIS requires that the underlying bus protocol support at least two concurrent bidirectional communication **channels** per device and guarantee in-order and reliable message delivery per channel. The bus protocol is also required to provide the flow control mechanism. The module corresponding to these tasks is the bus interface on both the device and the host.

The format of the actual networking data encapsulated in each packet depends on the underlying wire or wireless protocol such as [IEEE802.3-2008] or [IEEE802.11-2007].

1.5 Prerequisites/Preconditions

This protocol assumes that the device has already been successfully identified by the host operating system, and is ready to be set up for use.

This protocol also assumes that the device has already successfully identified what host operating system is in use on the host, and hence knows the relevant Object Identifiers (**OIDs**) and their semantics.

1.6 Applicability Statement

The protocol can be used with any external bus-based devices that have the ability to provide network access over a medium understood by the host operating system that supports sequential delivery, at least two concurrent bidirectional channels for exchange of control and data, and provides message boundaries so that this protocol can determine message sizes. .

1.7 Versioning and Capability Negotiation

Capability Negotiation: Determined by the highest-numbered protocol version, determined by the major version and minor version supported by both the host and the device. Furthermore, the host

queries the set of OIDs that the device supports on that host, at the time of initialization. See section 3.2.5.1 for details.

1.8 Vendor-Extensible Fields

The following fields are defined by the host operating system:

- The Object Identifiers (OIDs) and their semantics.<1>
- Status data (see section 2.2.7),<2>
- Out-of-band data type values (see section 2.2.14.1)<3>
- Per-packet data type values (see section 2.2.14.2).<4>

Status values defined in the common status values table specified in section 2.2.1.2 can be extended by the operating system vendor as appropriate.

No RNDIS fields can be directly extended by device vendors. However, a device vendor can supply additional configurable parameters for the device using an OID as described in the documentation of OID_GEN_RNDIS_CONFIG_PARAMETER in [MSDN-OIDs].

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

RNDIS messages are transported between the host and the device by the protocol; the host and the device are connected through by encapsulating them in the bus-native protocol message formats.<5> Messages that are sent on data and control channels MUST comply with the specification of the bus protocol they are mapped to respectively, including any **padding** requirements. The only RNDIS message that has explicit padding requirements is the REMOTE_NDIS_PACKET_MSG message, as defined in section 2.2.14.

2.2 Message Syntax

The REMOTE_NDIS_PACKET_MSG message is the only message exchanged over the data channel, and the rest of the messages defined in the following table are control messages. All multibyte values are represented in **little-endian byte order**.

2.2.1 Common Fields and Values

This section defines the header fields and their values that are common to multiple RNDIS messages.

2.2.1.1 RNDIS Message Types

The following table lists the message names, message type values, and descriptions.

RNDIS message types

Message Name	Message type value	Description
REMOTE_NDIS_PACKET_MSG	0x00000001	The host and device use this to send network data to one another.
REMOTE_NDIS_INITIALIZE_MSG	0x00000002	Sent by the host to initialize the device.
REMOTE_NDIS_INITIALIZE_CMPLT	0x80000002	Device response to an initialize message.
REMOTE_NDIS_HALT_MSG	0x00000003	Sent by the host to halt the device. This does not have a response. It is optional for the device to send this message to the host.
REMOTE_NDIS_QUERY_MSG	0x00000004	Sent by the host to send a query OID.
REMOTE_NDIS_QUERY_CMPLT	0x80000004	Device response to a query OID.
REMOTE_NDIS_SET_MSG	0x00000005	Sent by the host to send a set OID.
REMOTE_NDIS_SET_CMPLT	0x80000005	Device response to a set OID.
REMOTE_NDIS_RESET_MSG	0x00000006	Sent by the host to perform a soft reset on the device.
REMOTE_NDIS_RESET_CMPLT	0x80000006	Device response to reset message.
REMOTE_NDIS_INDICATE_STATUS_MSG	0x00000007	Sent by the device to indicate its status or an error when an unrecognized message is

Message Name	Message type value	Description
		received.
REMOTE_NDIS_KEEPALIVE_MSG	0x00000008	During idle periods, sent every few seconds by the host to check that the device is still responsive. It is optional for the device to send this message to check if the host is active.
REMOTE_NDIS_KEEPALIVE_CMPLT	0x80000008	The device response to a **keepalive** message. The host can respond with this message to a **keepalive** message from the device when the device implements the optional **KeepAliveTimer**.

2.2.1.2 Common Status Values

The following table shows the common status values along with descriptions.

Status values in RNDIS messages

Status identifier	Value	Description
RNDIS_STATUS_SUCCESS	0x00000000	Success
RNDIS_STATUS_FAILURE	0xC0000001	Unspecified error
RNDIS_STATUS_INVALID_DATA	0xC0010015	Invalid data error
RNDIS_STATUS_NOT_SUPPORTED	0xC00000BB	Unsupported request error
RNDIS_STATUS_MEDIA_CONNECT	0x4001000B	Device is connected to a network medium
RNDIS_STATUS_MEDIA_DISCONNECT	0x4001000C	Device is disconnected from the medium

2.2.2 REMOTE_NDIS_INITIALIZE_MSG

This message is sent by the host to initialize the device.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MessageType																															
MessageLength																															
RequestID																															
MajorVersion																															
MinorVersion																															

MaxTransferSize

MessageType (4 bytes): Identifies the type of the RNDIS message and MUST be set to 0x00000002.

MessageLength (4 bytes): The length of this message, in bytes. MUST be set to 0x00000018.

RequestID (4 bytes): A value, generated by the host, used to match the host's sent request to the response from the device. It is the responsibility of the host to ensure the uniqueness of this value among all outstanding request messages sent to the device.

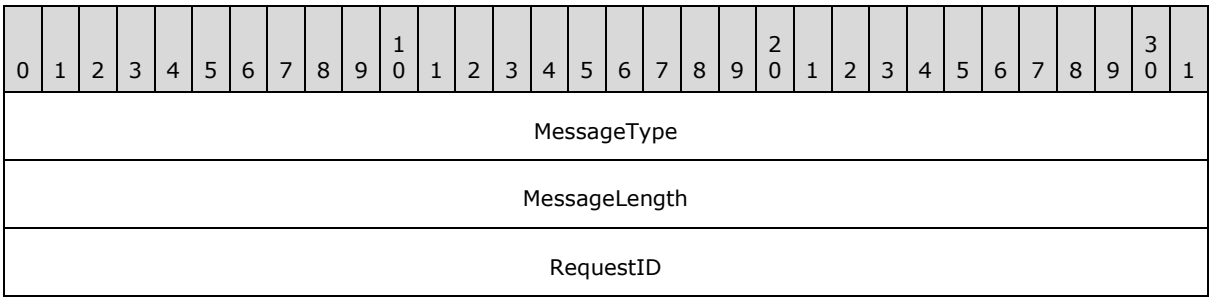
MajorVersion (4 bytes): The major version of the RNDIS protocol implemented by the host. It MUST be set to 0x00000001.

MinorVersion (4 bytes): The minor version of the RNDIS protocol implemented by the host. It MUST be set to 0x00000000.

MaxTransferSize (4 bytes): The maximum size, in bytes, of any single bus data transfer that the host expects to receive from the device. It SHOULD be set to 0x00004000.

2.2.3 REMOTE_NDIS_HALT_MSG

This message is sent by the host to halt the device.



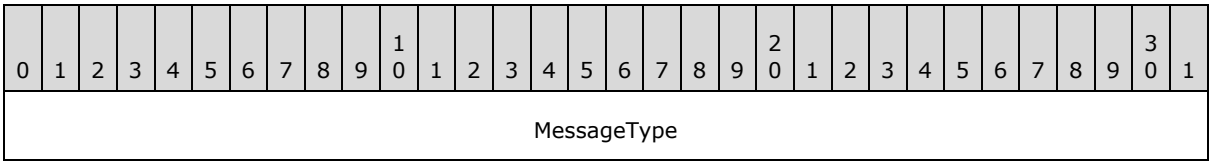
MessageType (4 bytes): Identifies the type of the RNDIS message and MUST be set to 0x00000003.

MessageLength (4 bytes): The length of this message, in bytes. It MUST be set to 0x0000000C.

RequestID (4 bytes): A value generated uniquely per device by the host to track this message.

2.2.4 REMOTE_NDIS_QUERY_MSG

This message is sent by the host to query an OID, using input data provided by the higher-layer protocol or application. The input data is opaque to this protocol and is simply passed through to the other end.



MessageLength
RequestID
Oid
InformationBufferLength
InformationBufferOffset
Reserved
OIDInputBuffer (variable)
...

MessageType (4 bytes): Identifies the type of the RNDIS message and MUST be set to 0x00000004.

MessageLength (4 bytes): The total length, in bytes, of this message, including the header and the **OIDInputBuffer**.

RequestID (4 bytes): A value, generated by the host, used to match the host's sent request to the response from the device. It is the responsibility of the host to ensure the uniqueness of this value among all outstanding request messages sent to the device.

Oid (4 bytes): The integer value of the OID, for the parameter of the device being queried for. Its value is defined by the higher-layer protocol or application.

InformationBufferLength (4 bytes): The length, in bytes, of the input data required for the OID query. This MUST be set to 0 when there is no input data associated with the OID.

InformationBufferOffset (4 bytes): The offset, in bytes, from the beginning of **RequestID** field where the input data for the query is located in the message. This value MUST be set to 0 when there is no input data associated with the OID.

Reserved (4 bytes): Reserved for future use and MUST be set to 0.

OIDInputBuffer (variable): The input data supplied by the host, required for the OID query request processing by the device, as per the host **NDIS** specification.

2.2.5 REMOTE_NDIS_SET_MSG

This message is sent by the host to set an OID, using input data provided by the higher-layer protocol or application. The input data is opaque to this protocol and is simply passed through to the other end.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MessageType																															

MessageLength
RequestID
Oid
InformationBufferLength
InformationBufferOffset
Reserved
OIDInputBuffer (variable)
...

MessageType (4 bytes): Identifies the type of the RNDIS message and MUST be set to 0x00000005.

MessageLength (4 bytes): The total length of this message, including the header and the **OIDInputBuffer**.

RequestID (4 bytes): A value generated by the host and used to match the host's sent request to the response from the device. When responding to this message from the host, the device MUST use this value in the **RequestID** field of the response message. It is the responsibility of the host to ensure the uniqueness of this value among all outstanding request messages sent to the device.

Oid (4 bytes): The integer value of the OID, uniquely identifying a parameter of the device that is being set. Its values are defined by the higher-layer application or protocol.

InformationBufferLength (4 bytes): The length, in bytes, of the input data required for the OID query. This must be set to 0x00000014 when there is input data associated with the OID, and MUST be set to 0 when there is no input data associated with the OID.

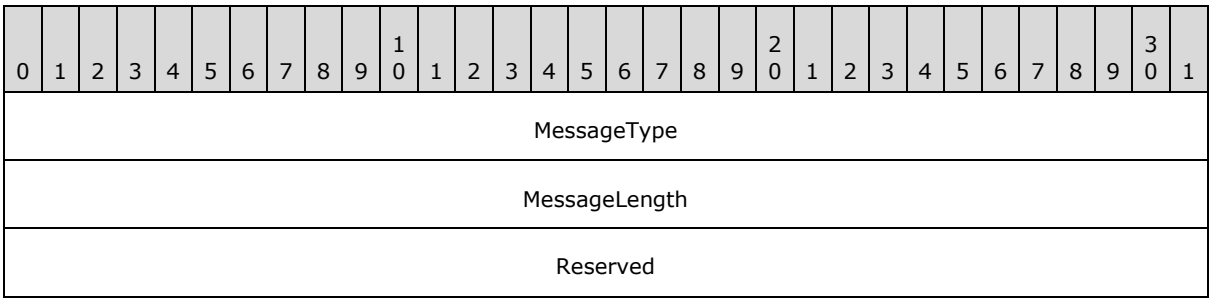
InformationBufferOffset (4 bytes): The offset, in bytes, from the beginning of the **RequestID** field, where the input data for the query is located in the message. This MUST be set to 0 when there is no input data associated with the OID.

Reserved (4 bytes): Reserved for future use and MUST be set to 0. The device MUST treat it as an error otherwise.

OIDInputBuffer (variable): The input data required for the OID set request to be processed by the device.

2.2.6 REMOTE_NDIS_RESET_MSG

This message is sent by the host to perform a soft reset on the device.



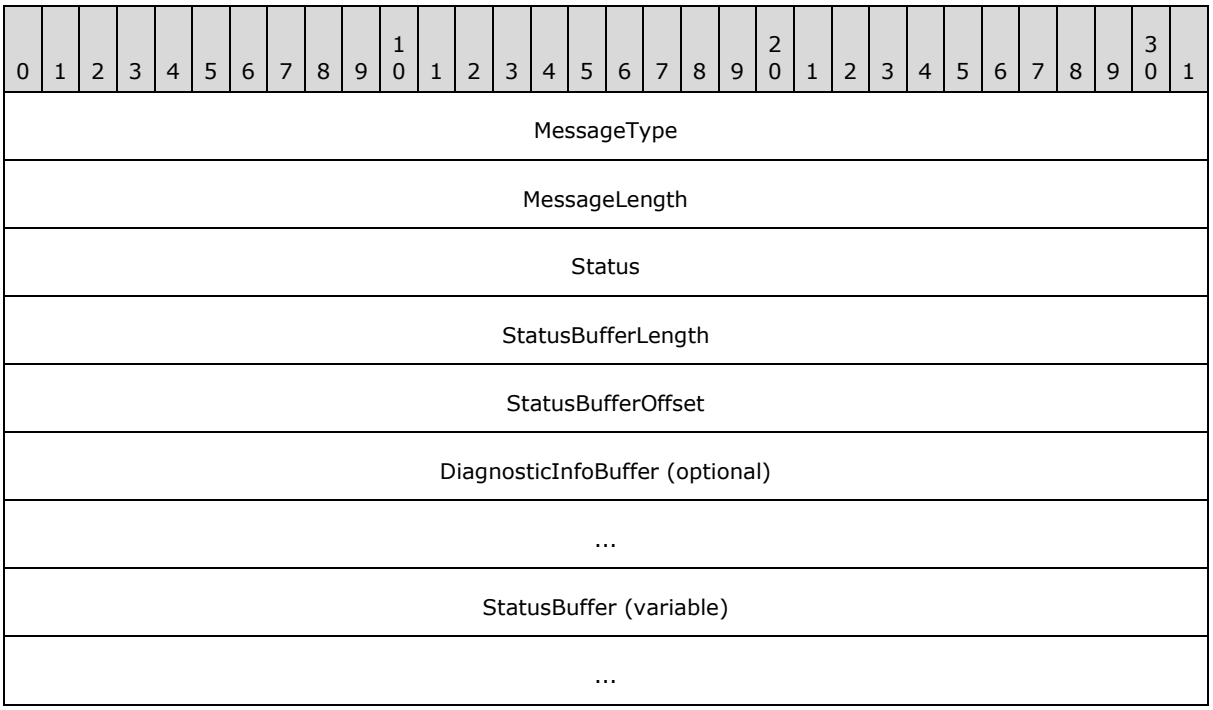
MessageType (4 bytes): Identifies the type of the RNDIS message and MUST be set to 0x00000006.

MessageLength (4 bytes): The total length of this message in bytes. MUST be set to 0x0000000C.

Reserved (4 bytes): Reserved for future use. MUST be set to zero.

2.2.7 REMOTE_NDIS_INDICATE_STATUS_MSG

This message is sent by the device to indicate its status or an error when an unrecognized message is received. The status data associated with a status change is provided by the higher-layer protocol or application. The status data is opaque to this protocol and is simply passed through to the higher layer on the host.



MessageType (4 bytes): Identifies the type of the RNDIS message and MUST be set to 0x00000007.

MessageLength (4 bytes): The total length of this message, in bytes, including the **StatusBuffer** field, if any.

Status (4 bytes): A value indicating a status change on the device, including link state change such as media connect/disconnect, or an error value when the device receives a malformed message. See the common status values table in section 2.2.1.2 for values used by the RNDIS protocol implementation itself. Other values of the **Status** field can be defined by the higher-layer protocol or application.

StatusBufferLength (4 bytes): The length of the **StatusBuffer** field, in bytes. This MUST be set to 0x00000000 when there is no **StatusBuffer**.<6>

StatusBufferOffset (4 bytes): When the **Status** field contains an error code, **StatusBufferOffset** is the offset, in bytes, from the beginning of the **Status** field at which the **DiagnosticInfoBuffer** field is located in the message. This buffer contains RNDIS_DIAGNOSTIC_INFO, as specified in section 2.2.7.1. In this case, the **StatusBuffer** contains the offending message that caused the device to send the REMOTE_NDIS_INDICATE_STATUS_MSG message to the host.

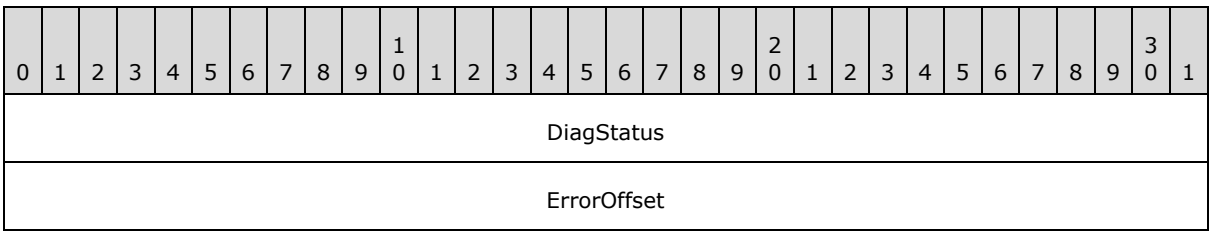
It is located immediately following the **DiagnosticInfoBuffer**. When the **Status** field represents a device state change such as a link state change, **StatusBufferOffset** is the offset, in bytes, from the beginning of the **Status** field at which the **StatusBuffer** is located. MUST be set to 0 if there is no associated status buffer.

DiagnosticInfoBuffer (8 bytes): This field is optional. **DiagnosticInfoBuffer** (RNDIS_DIAGNOSTIC_INFO) MUST be included in a response to a malformed message (that is, an unsupported message type) or when a REMOTE_NDIS_PACKET_MSG is received with inappropriate content, and the device cannot respond with any other RNDIS message. See section 2.2.7.1.

StatusBuffer (variable): This field is optional. When the **Status** field contains NDIS_STATUS_MEDIA_CONNECT, NDIS_STATUS_MEDIA_DISCONNECT, or NDIS_STATUS_LINK_SPEED_CHANGE, or any upper-layer defined value indicating a device state change, the StatusBuffer MUST contain any corresponding information for the specific status indication as required by the host<7>

If the status indication does not have any additional information, **StatusBuffer** is not present. In the case of a malformed message, **StatusBuffer** MUST contain the message that was malformed.

2.2.7.1 RNDIS_DIAGNOSTIC_INFO



DiagStatus (4 bytes): Contains additional status information about the error itself. The common status values table in section 2.2.1.2 lists the possible values.

ErrorOffset (4 bytes): Specifies the zero-based offset from the beginning of the offending message contained in the **StatusBuffer** , in bytes, at which the error was detected.

2.2.8 REMOTE_NDIS_KEEPALIVE_MSG

This message is sent by the host to check that device is still responsive.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MessageType																															
MessageLength																															
RequestID																															

MessageType (4 bytes): Identifies the type of the RNDIS message and MUST be set to 0x00000008.

MessageLength (4 bytes): The total length of this message, in bytes. MUST be set to 0x0000000C.

RequestID (4 bytes): A value, generated by the host, used to match the host sent request to the response from the device. It is the responsibility of the host to ensure the uniqueness of this value among all outstanding request messages sent to the device.

2.2.9 REMOTE_NDIS_INITIALIZE_CMPLT

This message is sent by the device in response to an initialize message.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MessageType																															
MessageLength																															
RequestID																															
Status																															
MajorVersion																															
MinorVersion																															
DeviceFlags																															
Medium																															
MaxPacketsPerTransfer																															
MaxTransferSize																															
PacketAlignmentFactor																															

Reserved
...

MessageType (4 bytes): Identifies the type of the RNDIS message type and MUST be set to 0x80000002.

MessageLength (4 bytes): The total length of this message, in bytes. MUST be set to 0x00000030.

RequestID (4 bytes): The **RequestID** field of the REMOTE_NDIS_INITIALIZE_MSG message to which this message is a response.

Status (4 bytes): The initialization status of the device. MUST be set to RNDIS_STATUS_SUCCESS (0x00000000) upon successful initialization or upon failure, and set to an appropriate error status value defined in the common status values table in section 2.2.1.2.

MajorVersion (4 bytes): Together with MinorVersion, the highest-numbered RNDIS Protocol version supported by the device. It MUST be set to 0x00000001.

MinorVersion (4 bytes): Together with **MajorVersion**, the highest-numbered RNDIS Protocol version supported by the device. It MUST be set to 0x00000000.

DeviceFlags (4 bytes): MUST be set to 0x000000010. Other values are reserved for future use.

Medium (4 bytes): Identifies the physical medium type. MUST be set to 0x000000000, which represents IEEE 802.3 wired Ethernet. Other values are reserved for future use.

MaxPacketsPerTransfer (4 bytes): The maximum number of concatenated REMOTE_NDIS_PACKET_MSG messages that the device can handle in a single bus transfer to it. This value MUST be at least 1.

MaxTransferSize (4 bytes): The maximum size, in bytes, of any single bus data transfer that the device expects to receive from the host. The value MUST be at least 0x0000002C (44 bytes) for the protocol to be operational with a **PacketAlignmentFactor** of 4.

PacketAlignmentFactor (4 bytes): The byte alignment the device expects for each RNDIS message that is part of a multimessage transfer to it. The value is specified as an exponent of 2; the host uses $2^{\{\textbf{PacketAlignmentFactor}\}}$ as the alignment value.

Reserved (8 bytes): Reserved for future use and MUST be set to zero. It SHOULD be treated as an error otherwise.

2.2.10 REMOTE_NDIS_QUERY_CMPLT

This message is sent by the device in response to a query OID message, with response data provided by the higher-layer protocol or application. The response data is opaque to this protocol and is simply passed through to the other end.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MessageType																															
MessageLength																															
RequestID																															
Status																															
InformationBufferLength																															
InformationBufferOffset																															
OIDInputBuffer (variable)																															
...																															

MessageType (4 bytes): Identifies the type of the RNDIS message and MUST be set to 0x80000004.

MessageLength (4 bytes): The total length of this message, in bytes, including the OIDInputBuffer, if any.

RequestID (4 bytes): The value in the **RequestID** field of the REMOTE_NDIS_QUERY_MSG message, to which this is a response.

Status (4 bytes): The status of processing for the query request. The common status values table in section 2.2.1.2 lists the allowed values and their meaning.

InformationBufferLength (4 bytes): The length, in bytes, of the data in the response to the query. This MUST be set to 0x00000000 when there is no OIDInputBuffer.

InformationBufferOffset (4 bytes): The offset, in bytes, from the beginning of **RequestID** field where the response data for the query is located in the message. This MUST be set to 0x00000000 when there is no **OIDInputBuffer**.

OIDInputBuffer (variable): The response data to the OID query request as specified by the host. **OIDInputBuffer** is not required when the OID specification does not require any information associated with the **Oid** field in REMOTE_NDIS_QUERY_MSG to which this is the response.

2.2.11 REMOTE_NDIS_SET_CMPLT

This message is sent by the device in response to a REMOTE_NDIS_SET_MSG message.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MessageType																															

MessageLength
RequestID
Status

MessageType (4 bytes): Identifies the type of the RNDIS message and MUST be set to 0x80000005.

MessageLength (4 bytes): The total length of this message, in bytes. MUST be set to 0x00000010.

RequestID (4 bytes): The value in the **RequestID** field of the REMOTE_NDIS_SET_MSG message, to which this is a response.

Status (4 bytes): The status of processing for the REMOTE_NDIS_SET_MSG message request. The common status values table in section 2.2.1.2 lists the allowed values and their meaning.

2.2.12 REMOTE_NDIS_RESET_CMPLT

This message is sent by the device in response to a REMOTE_NDIS_RESET_MSG message.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MessageType																															
MessageLength																															
Status																															
AddressingReset																															

MessageType (4 bytes): Identifies the type of the RNDIS message and MUST be set to 0x80000006.

MessageLength (4 bytes): The total length of this message, in bytes. MUST be set to 0x00000010.

Status (4 bytes): The status of processing for the REMOTE_NDIS_RESET_MSG message request by the device to which this message is the response. The common status values table in section 2.2.1.2 lists the allowed values and their meaning.

AddressingReset (4 bytes): This field indicates whether the addressing information, which is the multicast address list or packet filter, has been lost during the reset operation. This MUST be set to 0x00000001 if the device requires that the host to resend addressing information or MUST be set to zero otherwise.

2.2.13 REMOTE_NDIS_KEEPALIVE_CMPLT

This message is sent in response to a REMOTE_NDIS_KEEPALIVE_MSG message.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MessageType																															
MessageLength																															
RequestID																															
Status																															

MessageType (4 bytes): Identifies the type of the RNDIS message and MUST be set to 0x80000008.

MessageLength (4 bytes): The total length of this message, in bytes. MUST be set to 0x00000010.

RequestID (4 bytes): The value in the **RequestID** field of the REMOTE_NDIS_KEEPALIVE_MSG message, to which this message is a response.

Status (4 bytes): The status of processing for the REMOTE_NDIS_KEEPALIVE_MSG message request. The common status values table in section 2.2.1.2 lists the allowed values and their meaning.

2.2.14 REMOTE_NDIS_PACKET_MSG

This message is used by the host and the device to send packet data, out-of-band data, and per-packet info data to one another.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MessageType																															
MessageLength																															
DataOffset																															
DataLength																															
OutOfBandDataOffset																															
OutOfBandDataLength																															
NumOutOfBandDataElements																															
PerPacketInfoOffset																															
PerPacketInfoLength																															

Reserved
...
PayLoad (variable)
...
Padding (variable)
...
REMOTE_NDIS_PACKET_MSG (variable)
...

MessageType (4 bytes): Identifies the type of the RNDIS message and MUST be set to 0x00000001.

MessageLength (4 bytes): The total length of this RNDIS message including the header, payload, and padding.

DataOffset (4 bytes): The offset, in bytes, from the start of the **DataOffset** field of this message to the start of the packet data portion of the **PayLoad** field. This MUST be an integer multiple of 4.

DataLength (4 bytes): Specifies the number of bytes in the packet data of the **PayLoad** field of this message.

OutOfBandDataOffset (4 bytes): Specifies the offset, in bytes, of the first **out-of-band data** record in the **PayLoad** field, counting from the start of the **DataOffset** field in this message. MUST be an integer multiple of 4 when out-of-band data is present or set to 0x00000000 otherwise. When there are multiple out-of-band data records, each subsequent record MUST immediately follow the previous out-of-band data record.

OutOfBandDataLength (4 bytes): Specifies, in bytes, the total length of the out-of-band data.

NumOutOfBandDataElements (4 bytes): Specifies the number of out-of-band records in this message.

PerPacketInfoOffset (4 bytes): Specifies the offset, in bytes, of the start of per-packet-info data record (section 2.2.14.2) in the **PayLoad** field, counting from the start of the **DataOffset** field in this message. MUST be an integer multiple of 4 when **per-packet-info data** record is present or MUST be set to 0x00000000 otherwise. When there are multiple per-packet-info data records, each subsequent record MUST immediately follow the previous record.

PerPacketInfoLength (4 bytes): Specifies, in bytes, the total length of per-packet-information contained in this message.

Reserved (8 bytes): Reserved for future and MUST be set to zero. The host and the device MUST treat it as an error otherwise.

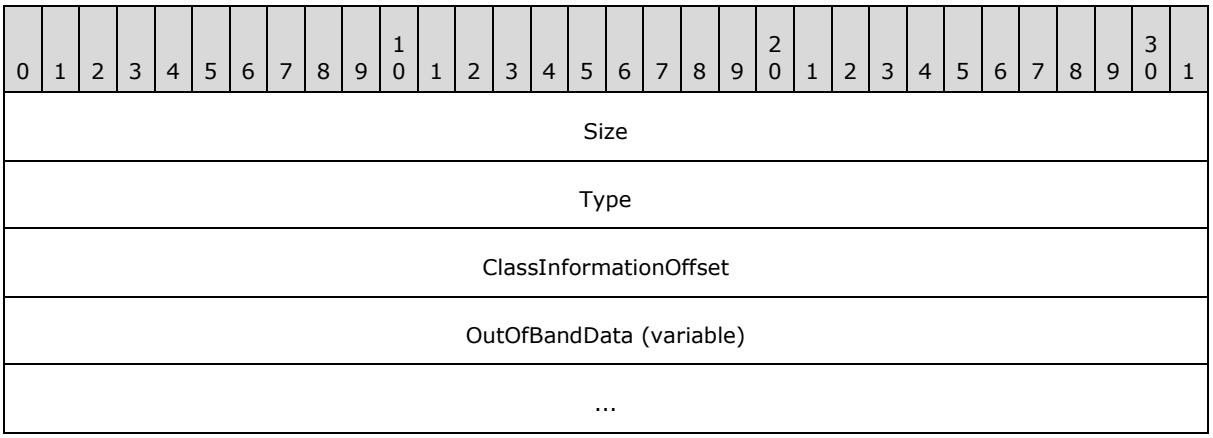
PayLoad (variable): Network data contained in this message. Specifically, the field consists of **DataLength** bytes of packet data, **OutOfBandDataLength** bytes of out-of-band data, and **PerPacketInfoLength** bytes of per-packet information, although those three sections can occur in any order.

Padding (variable): Additional bytes of zeros added at the end of the message to comply with the internal and external padding requirements. Internal padding MUST be as per the specification of the out-of-band data record and per-packet-info data record. The external padding size MUST be determined based on the **PacketAlignmentFactor** field specification in REMOTE_NDIS_INITIALIZE_CMPLT message by the device, when multiple **REMOTE_NDIS_PACKET_MSG** messages are bundled together in a single bus transfer. In this case, all but the very last **REMOTE_NDIS_PACKET_MSG** MUST respect the **PacketAlignmentFactor** field.

REMOTE_NDIS_PACKET_MSG (variable): This field is optional. When the host and device both support multiple packets per single bus transfer (see section 2.2.9), and multiple packets are available to send to the other side, multiple **REMOTE_NDIS_PACKET_MSG** messages will be appended in the same bus transfer for improved throughput. (This is solely for the purpose of improved performance.) Both the **MaxPacketsPerTransfer** and the **MaxTransferSize** values specified through the REMOTE_NDIS_INITIALIZE_CMPLT message MUST be respected.<8>

2.2.14.1 Out-of-Band Data Record

The out-of-band data record contains out-of-band data provided by the higher-layer protocol or application. The out-of-band data is opaque to this protocol and is simply passed through to the other end.



Size (4 bytes): Length, in bytes, of this header and appended out-of-band data and padding. This value MUST be an integer multiple of 4.

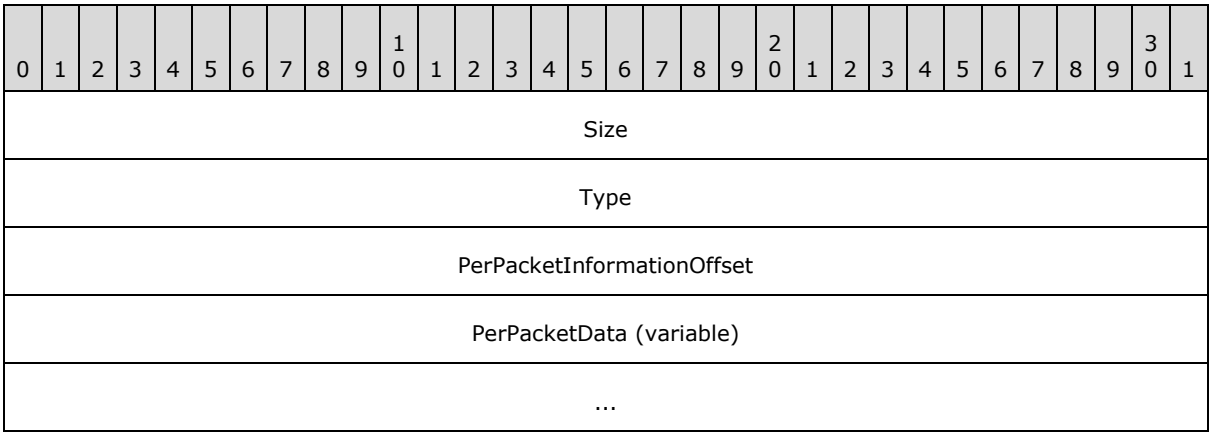
Type (4 bytes): MUST be as per host operating system specification.

ClassInformationOffset (4 bytes): The byte offset from the beginning of this out-of-band data record to the beginning of out-of-band data.

OutOfBandData (variable): The out-of-band data.

2.2.14.2 Per-Packet-Info Data Record

The per-packet-info data record contains per-packet data provided by the higher-layer protocol or application. The per-packet data is opaque to this protocol and is simply passed through to the other end.



Size (4 bytes): Length, in bytes, of this header plus **PerPacketData** plus any padding necessary. MUST be an integer multiple of 4.

Type (4 bytes): MUST be as per the host operating system specification.

PerPacketInformationOffset (4 bytes): The byte offset from the beginning of this per-packet-info record to the beginning of **PerPacketData**.

PerPacketData (variable): The per-packet data.

3 Protocol Details

3.1 Host Details

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. The document does not mandate that the implementations adhere to this model as long as their external behavior is consistent with that described in this document.

Following is the state diagram depicting various states the system could be in, what operations are initiated in each of the states, and what operations trigger a state transition.

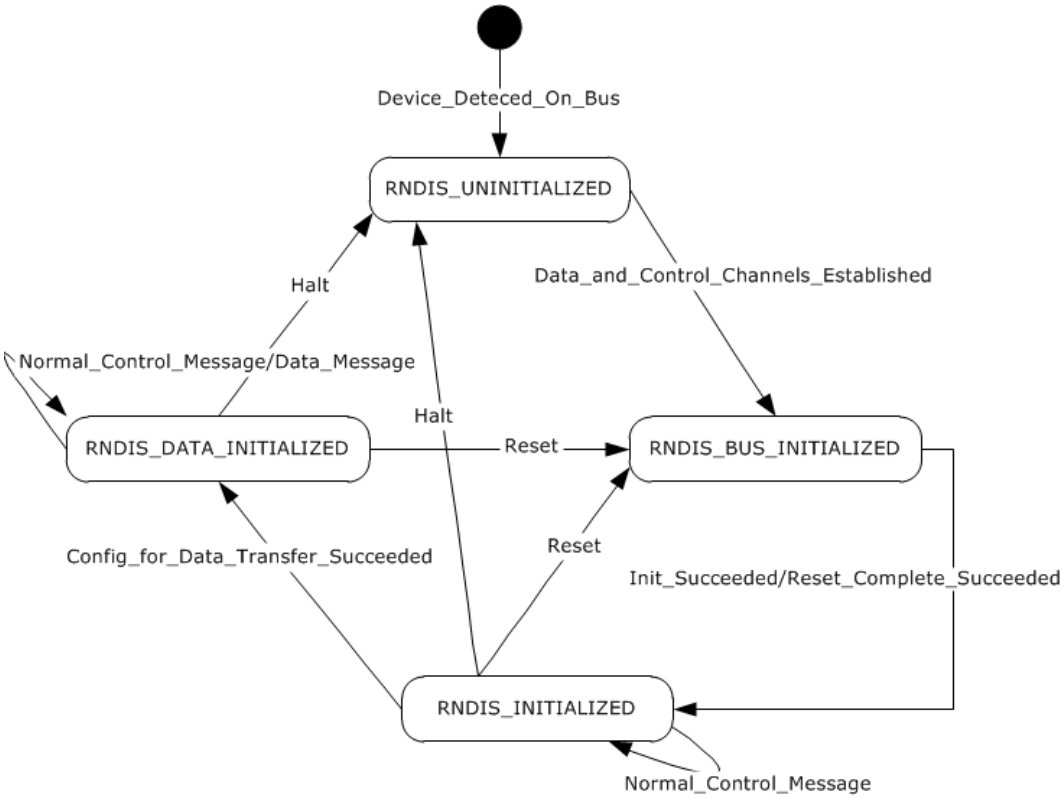


Figure 2: RNDIS Abstract Data Model State diagram

In addition to the protocol state described in the preceding figure, the following state variables are maintained by the host per device:

Effective Version: The highest matching MajorVersion and MinorVersion values of the RNDIS protocol used for communicating with the device.

Effective Max Transfer Size: The maximum size of the RNDIS message that can be exchanged between the host and the device.

Max Packets Per Transaction: The maximum number of network packets per bus transaction.

Byte Alignment: The byte alignment requirements in multi-packet transfers.

Media Type: The network media type of the device as per the NDIS specification on the host.

Outstanding Message: Either an empty message, or an outstanding control message sent to the device that has not yet received a corresponding response.

3.1.1.1 State Description

RNDIS_UNINITIALIZED: The host operating system is running, the device is connected to the host via the chosen bus transport, and the host is not yet configured to exchange any RNDIS messages.

RNDIS_BUS_INITIALIZED: The host has successfully mapped and initialized the RNDIS protocol data and control channels on the bus transport.

RNDIS_INITIALIZED: The host is configured to send and receive any of the RNDIS control messages for suitably configuring or querying the device, to receive status indications from the device, to reset the device, or to tear down the data and control channels.

RNDIS_DATA_INITIALIZED: This state is entered after the host has received REMOTE_NDIS_SET_CMPLT messages from the device in response to the REMOTE_NDIS_SET_MSG that it had sent earlier to the device with all the OIDs required to configure the device for data transfer. When the host is in this state, apart from the control messages, it can exchange REMOTE_NDIS_PACKET_MSG messages for network data transfer with the device on the data channel.

3.1.1.2 Transition Event Description

Device_Detected_On_Bus: This transition event occurs when the host detects the RNDIS device on the bus.

Data_and_Control_Channels_Established: This transition event occurs when the host successfully establishes the RNDIS abstract data and control channels with the device through a suitable mapping by using the bus transport.

Init_Succeeded: This transition occurs when the host receives a REMOTE_NDIS_INITIALIZE_CMPLT message on the control channel indicating success, in response to an earlier REMOTE_NDIS_INITIALIZE_MSG it had sent to the device. When this happens, the host MUST start the **KeepAlive Timer**.

Normal_Control_Message: This transition event occurs when the host receives a REMOTE_NDIS_QUERY_CMPLT, REMOTE_NDIS_SET_CMPLT, REMOTE_NDIS_INDICATE_STATUS_MSG, REMOTE_NDIS_KEEPALIVE_CMPLT or REMOTE_NDIS_KEEPALIVE_MSG message on the control channel.

Halt: This transition event occurs when the host receives a REMOTE_NDIS_HALT_MSG message from the device on the control channel or a bus-level disconnect or a hard-reset occurs. When this happens, the host MUST stop the **KeepAlive Timer**.

Reset: This transition event occurs when the host sends a REMOTE_NDIS_RESET_MSG message to the device. When this happens, the host MUST stop the **KeepAlive Timer**.

Reset_Complete_Succeeded: This transition event occurs when the host receives a REMOTE_NDIS_RESET_CMPLT message from the device, indicating success. When this happens, the host MUST start the **KeepAlive Timer**.

Config_For_Data_Transfer_Succeeded: This transition event occurs when the host discovers, via receipt of a REMOTE_NDIS_SET_CMPLT message, that all configuration required to initiate network data exchange with the device on the data channel has succeeded.

Data_Message: This transition event occurs when the host receives a REMOTE_NDIS_PACKET_MSG message from the device on the data channel.

3.1.2 Timers

KeepAliveTimer: Used by the host to determine the health of the device when the host and the device are in RNDIS_DATA_INITIALIZED state (section 3.1.1.1) and when there has been no other control or data traffic from the device to the host for a period chosen, based on the underlying bus characteristics. For a USB bus, the period SHOULD be 5 seconds.

ControlTimeoutTimer: Used by the host to impose a timeout, chosen based on the underlying bus characteristics, on the response from the device in return for any request sent on the control channel except for the REMOTE_NDIS_KEEPALIVE_MSG message. For a USB bus, the timeout SHOULD be 10 seconds.

3.1.3 Initialization

Host initialization begins when the device is detected by the host, and the host successfully establishes the data and control channels with the device and transitions to RNDIS_BUS_INITIALIZED state. The host then sends a REMOTE_NDIS_INITIALIZE_MSG message and processes the corresponding REMOTE_NDIS_INITIALIZE_CMPLT message as described in 3.1.5.1 and transitions to RNDIS_INITIALIZED state. If an error is encountered during initialization, the host MUST treat the device as unusable for network operations using the RNDIS protocol and appropriate steps to release any resources allocated MUST be taken.

3.1.4 Higher-Layer Triggered Events

Depending on the specification of the networking driver interface on the host operating system, the higher layers might trigger the host to send control or data messages to the device as specified in sections 3.1.1.1 and 3.1.1.2. The host and the device will be in one of the appropriate states as specified in section 3.1.1 as a result.

The host MUST serialize the sending of control messages to the device such that only one can be outstanding at a time.

3.1.5 Message Processing Events and Sequencing Rules

Whenever the host receives a message over the bus transport, the following tasks MUST be validated. Any errors encountered during the validation or the actual message processing MUST be deemed a protocol violation and indicate an error to the higher layer protocol or application. This higher layer protocol or application might choose to reset the device or halt it as deemed appropriate. If such a protocol violation is encountered, and the device has been initialized, then the host MUST send a REMOTE_NDIS_HALT_MSG or REMOTE_NDIS_RESET_MSG message as follows:

- Validate that the message size is not greater than the maximum RNDIS message size, and send a REMOTE_NDIS_HALT_MSG message if it is.
- Determine the type of the message and validate its minimum size and send a REMOTE_NDIS_HALT_MSG if not met.

- Validate that each of the fields contains a valid value and send a REMOTE_NDIS_RESET_MSG if not met
- Ensure that the host is in an appropriate state for receiving the message and send a REMOTE_NDIS_RESET_MSG if not.
- Ensure that the message was received on the correct channel (data channel or control channel) and send a REMOTE_NDIS_RESET_MSG if not
- If the message has a **RequestID** field, the **RequestID** MUST match the **RequestID** of the stored **Outstanding Message** that the host previously sent to the device.

The details of processing each specific message type are covered in the following sections, and in section 3.1.1.2.

3.1.5.1 Processing a REMOTE_NDIS_INITIALIZE_CMPLT Message

This message can be received only when the host and the device are in a **BUS_INITIALIZED** state. If the **Status** field of this message is set to **REMOTE_NDIS_STATUS_SUCCESS**, the host MUST move to the **RNDIS_INITIALIZED** state. For any other status value, the host MUST move to the **BUS_INITIALIZED** state. The host SHOULD move into the **UNINITIALIZED** state and release any resources acquired; the host MAY instead attempt to initialize the device again.

The following parameters MUST be derived by the host while processing this message, to be adhered to for the rest of the communication:

- The highest-numbered version of the RNDIS Protocol that the host and the device can support as described in section 1.7.
- The maximum size of any RNDIS message that the device and host can support together per bus transaction.
- The maximum number of network packets per bus transaction and byte alignment requirements.
- The network media type per the NDIS specification that the device supports.

3.1.5.2 Processing a REMOTE_NDIS_QUERY_CMPLT Message

If the **Status** field is set to RNDIS_STATUS_SUCCESS, then the device returned value MUST be read from the **InformationBuffer** field and MUST be indicated up to the higher layer. If the Status value is set to anything other than RNDIS_STATUS_SUCCESS, the Status MUST be indicated to the higher layer. This message can be received only when the host and the device are in the RNDIS_INITIALIZED or RNDIS_DATA_INITIALIZED states.

3.1.5.3 Processing a REMOTE_NDIS_SET_CMPLT Message

If the **Status** field is set to RNDIS_STATUS_SUCCESS, the corresponding set request as identified by the **RequestID** field is assumed to have succeeded. The **Status** value is indicated up to the higher layer. This message can be received only when the host is in the RNDIS_INITIALIZED or RNDIS_DATA_INITIALIZED states.

3.1.5.4 Processing a REMOTE_NDIS_KEEPALIVE_CMPLT Message

If the **Status** field is set to RNDIS_STATUS_SUCCESS, the KeepAliveTimer MUST be reset, and no other action is taken. Otherwise, the host sends a REMOTE_NDIS_RESET_MSG message to the

device. This can be received only when the host is in the RNDIS_DATA_INITIALIZED or RNDIS_INITIALIZED state.

3.1.5.5 Processing a REMOTE_NDIS_RESET_CMPLT Message

If the **Status** field is set to RNDIS_STATUS_SUCCESS, it indicates the device completed the soft-reset successfully, and the host MUST transition to the RNDIS_BUS_INITIALIZED state. If the **AddressingReset** field is set to 0x00000001, the host MUST indicate to the higher layers a need to configure the addressing information on the device again.

3.1.5.6 Processing a REMOTE_NDIS_INDICATE_STATUS_MSG Message

The reason for the device indicating the status can be derived by examining the **Status** field. If the value is RNDIS_STATUS_INVALID_DATA, an error MUST be indicated to the higher layer. If the value indicates a device status change, it MUST be indicated to the higher layer.

3.1.5.7 Processing a REMOTE_NDIS_HALT_MSG Message

If the host receives this message, irrespective of the value contained in the **RequestId** field, the host MUST free all the **resources** associated with this device, and the host MUST move to an **UNINITIALIZED** state. This message can be received only when the device and the host is in the **RNDIS_INITIALIZED** or **RNDIS_DATA_INITIALIZED** state.

3.1.5.8 Processing a REMOTE_NDIS_PACKET_MSG Message

When the host receives this message, the packet data contained in the message, along with any out-of-band data and per-packet-info data, MUST be indicated to the higher layer. The host MUST also examine if there is another REMOTE_NDIS_PACKET_MSG message immediately following this message, as multiple packets can be indicated together in a single bus message, and process the following message as if it is an independent data message until no more packets are found in the current bus message.

3.1.5.9 Processing a REMOTE_NDIS_KEEPALIVE_MSG Message

When the device implements the optional **KeepAliveTimer**, this message can be received when the host is in the **RNDIS_DATA_INITIALIZED** or **RNDIS_INITIALIZED** state and when there has not been any data or control message sent from host to device for the timeout period of the device's **KeepAliveTimer**. In response, the host MUST send a REMOTE_NDIS_KEEPALIVE_CMPLT message on the control channel with the **Status** field set to RNDIS_STATUS_SUCCESS.

3.1.6 Timer Events

KeepAliveTimer Expiry: The host MUST issue a REMOTE_NDIS_RESET_MSG message to the device when this timer expires.

ControlTimeoutTimer Expiry: The host MUST issue a REMOTE_NDIS_RESET_MSG message to the device when this timer expires.

3.1.7 Other Local Events

None.

3.2 Device Details

3.2.1 Abstract Data Model

The device maintains its protocol state, which is represented by the same diagram as used by the host in section 3.1.1.

In addition to the protocol state described in the preceding paragraph, following state variables are maintained by the host per device.

Effective Version: The highest matching MajorVersion and MinorVersion values of the RNDIS protocol used for communicating with the host.

Effective Max Transfer Size: The maximum size of the RNDIS message that can be exchanged between the host and the device.

3.2.1.1 State Description

RNDIS_UNINITIALIZED: The device is connected to the host on a bus but is not yet configured.

RNDIS_BUS_INITIALIZED: The RNDIS Protocol data and control channels are appropriately mapped and initialized on the device over the bus transport.

RNDIS_INITIALIZED: The device is ready to receive any of the RNDIS control messages for suitably configuring or querying the device, to indicate device status to the host, or to reset the protocol or tear down the communication.

RNDIS_DATA_INITIALIZED: This state is entered after the device has successfully processed all the REMOTE_NDIS_SET_MSG messages sent by the host with all the OIDs required to configure the device for data transfer, and has responded to the host with the appropriate REMOTE_NDIS_SET_CMPLT message. The device can now exchange any message allowed on the control channel, as well as the REMOTE_NDIS_PACKET_MSG messages for network data transfer on the data channel.

3.2.1.2 Transition Event Description

Device_Detected_On_Bus: This transition event occurs when the RNDIS device successfully initializes itself on the bus it is connected to on the host.

Data_and_Control_Channels_Established: This transition event occurs when the RNDIS abstract data and control channels are successfully established between the host and the device through a suitable mapping.

Init_Succeeded: This transition occurs when the device responds with a REMOTE_NDIS_INITIALIZE_CMPLT message to the host on the control channel in response to a REMOTE_NDIS_INITIALIZE_MSG message from the host. When this happens, the device MUST start the **KeepAlive Timer**, if one is implemented.

Normal_Control_Message: This transition event occurs when the device sends a REMOTE_NDIS_QUERY_CMPLT, REMOTE_NDIS_SET_CMPLT, REMOTE_NDIS_INDICATE_STATUS_MSG, REMOTE_NDIS_KEEPALIVE_CMPLT or REMOTE_NDIS_KEEPALIVE_MSG message to the host on the control channel.

Halt: This transition event occurs when the device sends or receives a REMOTE_NDIS_HALT_MSG message on the control channel or a bus-level disconnect or a hard-reset occurs. When this happens, the device MUST stop the **KeepAlive Timer**, if one is implemented.

Reset: This transition event occurs when the device receives a REMOTE_NDIS_RESET_MSG message from the host on the control channel. When this happens, the device MUST stop the **KeepAlive Timer**, if one is implemented.

Reset_Complete_Succeeded: This transition event occurs when the device sends a REMOTE_NDIS_RESET_CMPLT message to the host on the control channel. When this happens, the host MUST start the **KeepAlive Timer**, if one is implemented.

Config_For_Data_Transfer_Succeeded: This transition event occurs when any configuration required on the device side is successfully completed to initiate network data exchange between the device and the host.

Data_Message: This transition event occurs when device sends or receives a REMOTE_NDIS_PACKET_MSG message on the data channel.

3.2.2 Timers

KeepAliveTimer (optional): The device MAY implement a **KeepAliveTimer** to check whether the host is active, when the device is in the **RNDIS_DATA_INITIALIZED** or **RNDIS_INITIALIZED** state, and when there has been no other control or data traffic from the host for the timeout period. The timeout period should suitably be -chosen, based on the underlying bus characteristics.

3.2.3 Initialization

Device initialization happens in the context of processing a REMOTE_NDIS_INITIALIZE_MSG message. See section 3.2.5.1.

3.2.4 Higher-Layer Triggered Events

When the network link status is changed, the device MUST send a REMOTE_NDIS_INDICATE_STATUS_MSG message on the control channel.

When a networking packet arrives at the device to be sent to the host, the device MUST send a REMOTE_NDIS_PACKET_MSG message on the data channel.

3.2.5 Message Processing Events and Sequencing Rules

Whenever the device receives a message over the bus transport, the following tasks MUST be validated. Any errors encountered during the validation or the actual message processing MUST be deemed a protocol violation.

- Validate that the message size is not greater than the **Effective Max Transfer Size**.
- Determine the type of the message and validate its minimum size.
- Ensure that the message was received on the correct channel (data channel or control channel).

If the device receives a message that is not a valid RNDIS message, then it MUST send a REMOTE_NDIS_INDICATE_STATUS_MSG message on the control channel with an appropriate error code in the **Status** field, and MAY include a suitable RNDIS_DIAGNOSTIC_INFO message with the received message in the **StatusBuffer** field.

If the device is not in a state appropriate for the received message, or the device determines for any reason that it can no longer function and terminates the communication with the host, it SHOULD send a REMOTE_NDIS_HALT_MSG.

Except for a REMOTE_NDIS_HALT_MSG message, the device MUST copy the **RequestID** field of the request message to the response message that it sends to the host.

The details of processing each specific message type are covered in the following sections, and in section 3.2.1.2.

3.2.5.1 Processing a REMOTE_NDIS_INITIALIZE_MSG Message

This message is received only when the device and the host are in the **RNDIS_UNINITIALIZED** state. The device MUST update the **Effective MaxTransferSize** field based on the **MaxTransferSize** field value specified in this message in the subsequent communication with the host, and respond with a REMOTE_NDIS_INITIALIZE_CMPLT message. The highest protocol version that is supported by both the host and the device is used for the rest of the communication between them, as determined by the highest matching value of the mutually supported **MajorVersion** field followed by the highest value of the mutually supported **MinorVersion** field.

3.2.5.2 Processing a REMOTE_NDIS_QUERY_MSG message

This message can be received when the device is in the **RNDIS_INITIALIZED** or **RNDIS_DATA_INITIALIZED** state. If the device supports the OID contained in the message, it MUST respond with the appropriate REMOTE_NDIS_QUERY_CMPLT message, indicating the status of REMOTE_NDIS_QUERY_MSG request processing. Any input parameters required to service the OID MUST be read from the **InformationBuffer** field. Upon successful processing of the OID, the response data SHOULD be set in the **InformationBuffer** field of the REMOTE_NDIS_QUERY_CMPLT response message. If the OID is not supported by the higher layer on the device, the **Status** field of the REMOTE_NDIS_QUERY_CMPLT response message MUST be set to the value RNDIS_STATUS_NOT_SUPPORTED.

3.2.5.3 Processing REMOTE_NDIS_SET_MSG message

This message can be received by the device when the device is in **RNDIS_INITIALIZED** or **RNDIS_DATA_INITIALIZED** state. If the OID contained in the message is supported by the higher layer on the device, it MUST respond with a REMOTE_NDIS_SET_CMPLT message, indicating the status of processing the request. Any input parameters required to service the OID MUST be read from the **InformationBuffer** field. If the OID is not supported by the higher layer on the device, the **Status** field of the REMOTE_NDIS_SET_CMPLT response message MUST be set to the value RNDIS_STATUS_NOT_SUPPORTED.

3.2.5.4 Processing a REMOTE_NDIS_KEEPALIVE_MSG Message

This message can be received by the device when the device is in a **RNDIS_DATA_INITIALIZED** or **RNDIS_DATA_INITIALIZED** state when this message is received. The device MUST respond with a REMOTE_NDIS_KEEPALIVE_CMPLT message with the **Status** field value set to RNDIS_STATUS_SUCCESS if the device is working correctly or an error value to obtain a REMOTE_NDIS_RESET_MSG message from the host.

3.2.5.5 Processing a REMOTE_NDIS_RESET_MSG Message

This message can be received in either the **RNDIS_INITIALIZED** or **RNDIS_DATA_INITIALIZED** state. The device MUST reset its internal state and keep the data and control channels intact. The device MUST then respond to the host with a REMOTE_NDIS_RESET_CMPLT message containing the appropriate status.

3.2.5.6 Processing a REMOTE_NDIS_HALT_MSG Message

This message can be received only when the device is in the **RNDIS_INITIALIZED** or **RNDIS_DATA_INITIALIZED** state. The device MUST close the data and control channels with the host upon receiving this message. The device will not send any response to this message from the host, on processing this request. The device MUST then enter the **RNDIS_UNINITIALIZED** state.

3.2.5.7 Processing a REMOTE_NDIS_PACKET_MSG Message

When the device receives this message on the data channel, the packet data contained in the message, along with any out-of-band data and per-packet-info data, MUST be delivered to the higher layer. The device MUST also examine if there is another REMOTE_NDIS_PACKET_MSG message immediately following this message, as multiple REMOTE_NDIS_PACKET_MSG messages can be bundled in a single bus transfer, and process the following message as if it is an independent data message, until no more packets are found in the current bus message.

When the host receives this message, it processes it in a similar fashion and indicate the packet up to the higher layers.

3.2.5.8 Processing a REMOTE_NDIS_KEEPALIVE_CMPLT Message

When the device implements the optional KeepAliveTimer, the device MUST reset its KeepAliveTimer in response to this message.

3.2.6 Timer Events

KeepAliveTimer Expiry: When the KeepAliveTimer expires, the device MUST send a REMOTE_NDIS_KEEPALIVE_MSG message to the host via the control channel to check if the host is active; in the **RNDIS_DATA_INITIALIZED** or **RNDIS_INITIALIZED** states if there has been no other control or data traffic from the host for the **KeepAliveTimer's** timeout period. The **KeepAliveTimer's** timeout period is dependent on the underlying bus.

3.2.7 Other Local Events

Device status change: Whenever there is a change in the state of the device, such as a link state change, the device MUST indicate this change to the host by sending an appropriate REMOTE_NDIS_INDICATE_STATUS_MSG message over the control channel.

4 Protocol Examples

4.1 Example of Initialization

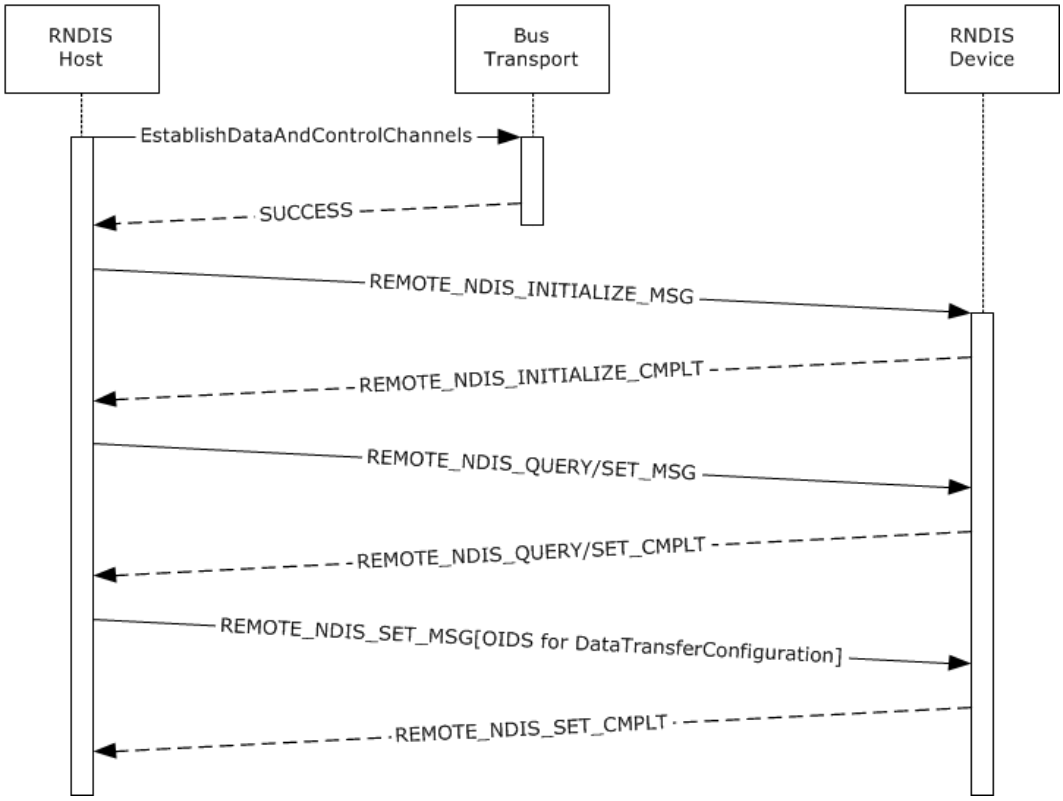


Figure 3: RNDIS Protocol initialization sequence diagram

The preceding figure illustrates the steps for a successful initialization.

1. The initialization occurs when the device is detected by the host operating system on a supported bus, represented by the UNINITIALIZED state.

2. Upon successful establishment of the data and control channels over the supported bus, from the BUS_INITIALIZED state, the host sends a REMOTE_NDIS_INITIALIZE_MSG message to the device. The device responds with a REMOTE_NDIS_INITIALIZE_CMPLT message specifying its basic capabilities. If this step fails, the device will be considered unusable, and the host cannot use it for any kind of network operations.

3. The host and the device are both in the RNDIS_INITIALIZED state and a sequence of REMOTE_NDIS_QUERY_MSG and REMOTE_NDIS_SET_MSG control messages are sent by the host to configure the device further, as specified in section 3.2.5. The device responds to these messages appropriately as specified in section 3.2.5.

4. After the device is configured, the host sends a REMOTE_NDIS_SET_MSG message with appropriate OIDs to initialize the data transfer. On a successful REMOTE_NDIS_SET_CMPLT response from the device, the host and the device enter the RNDIS_DATA_INITIALIZED state

and are ready for network data transfer. These OIDs can include parameters for configuring the device to indicate multicast, broadcast, or directed packets to the host.

4.2 Example of an OID Query and its Response

In the following example, the higher layer has defined an OID named _OID_GEN_MEDIA_SUPPORTED(0x0000ABCD) to query the list of media the device supports, which can be a subset of the values {Medium802_3 (0x00000000), Medium802_5 (0x00000001), MediumWan (0x00000002), and MediumFDDI (0x00000003)}.

The device's higher layer is expected to return the supported media in the **InformationBuffer** field as an array of 32-bit values. Assuming that the device supports Medium802_3, the following packet diagrams represent the REMOTE_NDIS_QUERY_MSG message from the host on the control channel and the corresponding successful REMOTE_NDIS_QUERY_CMPLT message from the device, respectively.

REMOTE_NDIS_QUERY_MSG for OID_GEN_MEDIA_SUPPORTED

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MessageType=0x00000004																															
MessageLength=0x0000001C																															
RequestID=0x0000012																															
OID=0x0000ABCD																															
InformationBufferLength=0x00000000																															
InformationBufferOffset=0x00000000																															
Reserved=0x00000000																															

REMOTE_NDIS_QUERY_CMPLT for OID_GEN_MEDIA_SUPPORTED

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
MessageType=0x80000004																															
MessageLength=0x0000001C																															
RequestID=0x00000012																															
Status=0x00000000																															
InformationBufferLength=0x00000004																															

InformationBufferOffset=0x00000010
0x00000000

4.3 Example of a Multi-Packet Message

The following example shows the structure of a REMOTE_NDIS_PACKET_MSG message on the data channel containing two packets, the first with a payload of 30 bytes and the second with a payload of 20 bytes. During initialization, through a REMOTE_NDIS_INITIALIZE_CMPLT message, the device specified the values of **MaxPacketsPerTransfer** to be 0x00000004, **MaxTransferSize** to be 0x00001000 (4096), and **PacketAlignmentFactor** to be 0x00000004. The **PacketAlignmentFactor** field indicates that 16-byte alignment is required between the multiple-packets, excluding the last, which requires 6-byte padding at the end of the first REMOTE_NDIS_PACKET_MSG message in the example.

Multi-packet REMOTE_NDIS_PACKET_MSG

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MessageType=0x00000001																															
MessageLength=0x00000050																															
DataOffset=0x00000024																															
DataLength=0x0000001E																															
OutOfBandDataOffset=0x00000000																															
OutOfBandDataLength=0x00000000																															
NumOutOfBandDataElements=0x00000000																															
PerPacketInfoOffset=0x00000000																															
PerPacketInfoLength=0x00000000																															
Reserved(low 32 bit)=0x00000000																															
Reserved(high 32 bit)=0x00000000																															
Payload(30 bytes)																															
Padding(6 bytes)																															
MessageType=0x00000001																															

MessageLength=0x00000040
DataOffset=0x00000024
DataLength=0x00000014
OutOfBandDataOffset=0x00000000
OutOfBandDataLength=0x00000000
NumOutOfBandDataElements=0x00000000
PerPacketInfoOffset=0x00000000
PerPacketInfoLength=0x00000000
Reserved(low 32 bit)=0x00000000
Reserved(high 32 bit)=0x00000000
Payload(20 bytes)

4.4 Example of RNDIS over USB

For an example of how RNDIS is mapped over the USB protocol, see [MSDN-RNDISUSB].

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Windows 2000 operating system
- Windows XP operating system
- Windows Server 2003 operating system
- Windows Vista operating system
- Windows Server 2008 operating system
- Windows 7 operating system
- Windows Server 2008 R2 operating system
- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 1.8: For a list of OIDs defined in Windows, with their associated input data structures and lengths, see [MSDN-OIDs].

<2> Section 1.8: For the structure to be used for a specific status indication for device state change in Windows, see [MSDN-OIDs].

<3> Section 1.8: For information about specific details in Windows, see the documentation of the NDIS_PACKET_OOB_DATA structure in [MSDN-NDIS].

<4> Section 1.8: For information about specific details in Windows, see the documentation of the NDIS_PACKET_EXTENSION structure in [MSDN-NDIS].

<5> Section 2.1: All versions of Windows support the following underlying bus transports: USB 1.0, USB 1.1, and USB 2.0. See [USB-SPC] for details.

<6> Section 2.2.7: See [MSDN-NDIS] for the value to be used for a specific status indication for device state change in Windows.

<7> Section 2.2.7: See [MSDN-NDIS] for the structure to be used for a specific status indication for device state change in Windows.

<8> Section 2.2.14: All Windows RNDIS implementations support sending and receiving multiple **REMOTE_NDIS_PACKET_MSG** messages in one bus transfer.

7 Change Tracking

This section identifies changes that were made to the [MS-RNDIS] protocol document between the February 2014 and May 2014 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- The removal of a document from the documentation set.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the technical content of the document is identical to the last released version.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.
- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.

- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
	Revised document for technical clarity and completeness.	Y	Content updated.

8 Index

V