CybOXTM Version 2.1.1 Part 38: Network Packet Object

Working Draft 01

15 December 2015

Technical Committee:

[OASIS Cyber Threat Intelligence (CTI) TC](https://www.oasis-open.org/committees/cti)

Chair:

Richard Struse ([Richard.Struse@HQ.DHS.GOV](mailto:Richard.Struse@HQ.DHS.GOV)), [DHS Office of Cybersecurity and Communications (CS&C)](http://www.dhs.gov/office-cybersecurity-and-communications)

Editors:

Desiree Beck ([dbeck@mitre.org](mailto:ikirillov@mitre.org)), [MITRE Corporation](http://www.mitre.org/)

Trey Darley ([trey@soltra.com](mailto:trey@soltra.com)), [Soltra](http://www.soltra.com/)

Ivan Kirillov ([ikirillov@mitre.org](mailto:ikirillov@mitre.org)), [MITRE Corporation](http://www.mitre.org/)

Rich Piazza ([rpiazza@mitre.org](mailto:ikirillov@mitre.org)), [MITRE Corporation](http://www.mitre.org/)

Additional artifacts:

This prose specification is one component of a Work Product which consists of:

* *CybOX™ Version 2.1.1 Part 01: Overview*. [URI]
* *CybOX™ Version 2.1.1 Part 02: Common*. [URI]
* *CybOX™ Version 2.1.1 Part 03: Core*. [URI]
* *CybOX™ Version 2.1.1 Part 04: Default Extensions*. [URI]
* *CybOX™ Version 2.1.1 Part 05: Default Vocabularies*. [URI]
* *CybOX™ Version 2.1.1 Part 06: UML Model*. [URI]
* *CybOX™ Version 2.1.1 Part 07: API Object*. [URI]
* *CybOX™ Version 2.1.1 Part 08: ARP Cache Object*. [URI]
* *CybOX™ Version 2.1.1 Part 09: AS Object*. [URI]
* *CybOXTM Version 2.1.1 Part 10: Account Object*. [URI]
* *CybOXTM Version 2.1.1 Part 11: Address Object*. [URI]
* *CybOXTM Version 2.1.1 Part 12: Archive File Object*. [URI]
* *CybOXTM Version 2.1.1 Part 13: Artifact Object*. [URI]
* *CybOXTM Version 2.1.1 Part 14: Code Object*. [URI]
* *CybOXTM Version 2.1.1 Part 15: Custom Object*. [URI]
* *CybOXTM Version 2.1.1 Part 16: DNS Cache Object*. [URI]
* *CybOXTM Version 2.1.1 Part 17: DNS Query Object*. [URI]
* *CybOXTM Version 2.1.1 Part 18: DNS Record Object*. [URI]
* *CybOXTM Version 2.1.1 Part 19: Device Object*. [URI]
* *CybOXTM Version 2.1.1 Part 20: Disk Object*. [URI]
* *CybOXTM Version 2.1.1 Part 21: Disk Partition Object*. [URI]
* *CybOXTM Version 2.1.1 Part 22: Domain Name Object*. [URI]
* *CybOXTM Version 2.1.1 Part 23: Email Message Object*. [URI]
* *CybOXTM Version 2.1.1 Part 24: File Object*. [URI]
* *CybOXTM Version 2.1.1 Part 25: GUI Dialogbox Object*. [URI]
* *CybOXTM Version 2.1.1 Part 26: GUI Object*. [URI]
* *CybOXTM Version 2.1.1 Part 27: GUI Window Object*. [URI]
* *CybOXTM Version 2.1.1 Part 28: HTTP Session Object*. [URI]
* *CybOXTM Version 2.1.1 Part 29: Hostname Object*. [URI]
* *CybOXTM Version 2.1.1 Part 30: Image File Object*. [URI]
* *CybOXTM Version 2.1.1 Part 31: Library File Object*. [URI]
* *CybOXTM Version 2.1.1 Part 32: Link Object*. [URI]
* *CybOXTM Version 2.1.1 Part 33: Linux Package Object*. [URI]
* *CybOXTM Version 2.1.1 Part 34: Memory Object*. [URI]
* *CybOXTM Version 2.1.1 Part 35: Mutex Object*. [URI]
* *CybOXTM Version 2.1.1 Part 36: Network Connection Object*. [URI]
* *CybOXTM Version 2.1.1 Part 37: Network Flow Object*. [URI]
* *CybOXTM Version 2.1.1 Part 38: Network Packet Object*. (this document)
* *CybOXTM Version 2.1.1 Part 39: Network Route Entry Object*. [URI]
* *CybOXTM Version 2.1.1 Part 40: Network Route Object*. [URI]
* *CybOXTM Version 2.1.1 Part 41: Network Socket Object*. [URI]
* *CybOXTM Version 2.1.1 Part 42: Network Subnet Object*. [URI]
* *CybOXTM Version 2.1.1 Part 43: PDF File Object*. [URI]
* *CybOXTM Version 2.1.1 Part 44: Pipe Object*. [URI]
* *CybOXTM Version 2.1.1 Part 45: Port Object*. [URI]
* *CybOXTM Version 2.1.1 Part 46: Process Object*. [URI]
* *CybOXTM Version 2.1.1 Part 47: Product Object*. [URI]
* *CybOXTM Version 2.1.1 Part 48: SMS Message Object*. [URI]
* *CybOXTM Version 2.1.1 Part 49: Semaphore Object*. [URI]
* *CybOXTM Version 2.1.1 Part 50: Socket Address Object*. [URI]
* *CybOXTM Version 2.1.1 Part 51: System Object*. [URI]
* *CybOXTM Version 2.1.1 Part 52: URI Object*. [URI]
* *CybOXTM Version 2.1.1 Part 53: URL History Object*. [URI]
* *CybOXTM Version 2.1.1 Part 54: Unix File Object*. [URI]
* *CybOXTM Version 2.1.1 Part 55: Unix Network Route Entry Object*. [URI]
* *CybOXTM Version 2.1.1 Part 56: Unix Pipe Object*. [URI]
* *CybOXTM Version 2.1.1 Part 57: Unix Process Object*. [URI]
* *CybOXTM Version 2.1.1 Part 58: Unix User Account Object*. [URI]
* *CybOXTM Version 2.1.1 Part 59: Unix Volume Object*. [URI]
* *CybOXTM Version 2.1.1 Part 60: User Account Object*. [URI]
* *CybOXTM Version 2.1.1 Part 61: User Session Object*. [URI]
* *CybOXTM Version 2.1.1 Part 62: Volume Object*. [URI]
* *CybOXTM Version 2.1.1 Part 63: Whois Object*. [URI]
* *CybOXTM Version 2.1.1 Part 64: Win Computer Account Object*. [URI]
* *CybOXTM Version 2.1.1 Part 65: Win Critical Section Object*. [URI]
* *CybOXTM Version 2.1.1 Part 66: Win Driver Object*. [URI]
* *CybOXTM Version 2.1.1 Part 67: Win Event Log Object*. [URI]
* *CybOXTM Version 2.1.1 Part 68: Win Event Object*. [URI]
* *CybOXTM Version 2.1.1 Part 69: Win Executable File Object*. [URI]
* *CybOXTM Version 2.1.1 Part 70: Win File Object*. [URI]
* *CybOXTM Version 2.1.1 Part 71: Win Filemapping Object*. [URI]
* *CybOXTM Version 2.1.1 Part 72: Win Handle Object*. [URI]
* *CybOXTM Version 2.1.1 Part 73: Win Hook Object*. [URI]
* *CybOXTM Version 2.1.1 Part 74: Win Kernel Hook Object*. [URI]
* *CybOXTM Version 2.1.1 Part 75: Win Kernel Object*. [URI]
* *CybOXTM Version 2.1.1 Part 76: Win Mailslot Object*. [URI]
* *CybOXTM Version 2.1.1 Part 77: Win Memory Page Region Object*. [URI]
* *CybOXTM Version 2.1.1 Part 78: Win Mutex Object*. [URI]
* *CybOXTM Version 2.1.1 Part 79: Win Network Route Entry Object*. [URI]
* *CybOXTM Version 2.1.1 Part 80: Win Network Share Object*. [URI]
* *CybOXTM Version 2.1.1 Part 81: Win Pipe Object*. [URI]
* *CybOXTM Version 2.1.1 Part 82: Win Prefetch Object*. [URI]
* *CybOXTM Version 2.1.1 Part 83: Win Process Object*. [URI]
* *CybOXTM Version 2.1.1 Part 84: Win Registry Key Object*. [URI]
* *CybOXTM Version 2.1.1 Part 85: Win Semaphore Object*. [URI]
* *CybOXTM Version 2.1.1 Part 86: Win Service Object*. [URI]
* *CybOXTM Version 2.1.1 Part 87: Win System Object*. [URI]
* *CybOXTM Version 2.1.1 Part 88: Win System Restore Object*. [URI]
* *CybOXTM Version 2.1.1 Part 89: Win Task Object*. [URI]
* *CybOXTM Version 2.1.1 Part 90: Win Thread Object*. [URI]
* *CybOXTM Version 2.1.1 Part 91: Win User Account Object*. [URI]
* *CybOXTM Version 2.1.1 Part 92: Win Volume Object*. [URI]
* *CybOXTM Version 2.1.1 Part 93: Win Waitable Timer Object*. [URI]
* *CybOXTM Version 2.1.1 Part 94: X509 Certificate Object*. [URI]

Related work:

This specification is related to:

* *STIXTM Version 1.2.1 (placeholder)*

Abstract:

The Cyber Observable Expression (CybOX) is a standardized language for encoding and communicating high-fidelity information about cyber observables, whether dynamic events or stateful measures that are observable in the operational cyber domain. By specifying a common structured schematic mechanism for these cyber observables, the intent is to enable the potential for detailed automatable sharing, mapping, detection and analysis heuristics. This specification document defines the Network Packet Object data model, which is one of the Object data models for CybOX content.

Status:

This [Working Draft](https://www.oasis-open.org/policies-guidelines/tc-process#dWorkingDraft) (WD) has been produced by one or more TC Members; it has not yet been voted on by the TC or [approved](https://www.oasis-open.org/policies-guidelines/tc-process#committeeDraft) as a Committee Draft (Committee Specification Draft or a Committee Note Draft). The OASIS document [Approval Process](https://www.oasis-open.org/policies-guidelines/tc-process#standApprovProcess) begins officially with a TC vote to approve a WD as a Committee Draft. A TC may approve a Working Draft, revise it, and re-approve it any number of times as a Committee Draft.

URI patterns:

Initial publication URI:  
http://docs.oasis-open.org/cti/stix/v1.2.1/csd01/part1-overview/stix-v1.2.1-csd01-part1-overview.docx

Permanent “Latest version” URI:  
http://docs.oasis-open.org/cti/stix/v1.2.1/stix-v1.2.1-part1-overview.docx

(Managed by OASIS TC Administration; please don’t modify.)

Copyright © OASIS Open 2016. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full [Policy](https://www.oasis-open.org/policies-guidelines/ipr) may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Portions copyright © United States Government 2012-2016.  All Rights Reserved.  
  
STIX™, TAXII™, AND CybOX™ (STANDARD OR STANDARDS) AND THEIR COMPONENT PARTS ARE PROVIDED “AS IS” WITHOUT ANY WARRANTY OF ANY KIND, EITHER EXPRESSED, IMPLIED, OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY THAT THESE STANDARDS OR ANY OF THEIR COMPONENT PARTS WILL CONFORM TO SPECIFICATIONS, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR FREEDOM FROM INFRINGEMENT, ANY WARRANTY THAT THE STANDARDS OR THEIR COMPONENT PARTS WILL BE ERROR FREE, OR ANY WARRANTY THAT THE DOCUMENTATION, IF PROVIDED, WILL CONFORM TO THE STANDARDS OR THEIR COMPONENT PARTS. IN NO EVENT SHALL THE UNITED STATES GOVERNMENT OR ITS CONTRACTORS OR SUBCONTRACTORS BE LIABLE FOR ANY DAMAGES, INCLUDING, BUT NOT LIMITED TO, DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES, ARISING OUT OF, RESULTING FROM, OR IN ANY WAY CONNECTED WITH THESE STANDARDS OR THEIR COMPONENT PARTS OR ANY PROVIDED DOCUMENTATION, WHETHER OR NOT BASED UPON WARRANTY, CONTRACT, TORT, OR OTHERWISE, WHETHER OR NOT INJURY WAS SUSTAINED BY PERSONS OR PROPERTY OR OTHERWISE, AND WHETHER OR NOT LOSS WAS SUSTAINED FROM, OR AROSE OUT OF THE RESULTS OF, OR USE OF, THE STANDARDS, THEIR COMPONENT PARTS, AND ANY PROVIDED DOCUMENTATION. THE UNITED STATES GOVERNMENT DISCLAIMS ALL WARRANTIES AND LIABILITIES REGARDING THE STANDARDS OR THEIR COMPONENT PARTS ATTRIBUTABLE TO ANY THIRD PARTY, IF PRESENT IN THE STANDARDS OR THEIR COMPONENT PARTS AND DISTRIBUTES IT OR THEM “AS IS.”

Table of Contents

[1 Introduction 8](#_Toc449967588)

[1.1 CybOXTM Specification Documents 8](#_Toc449967589)

[1.2 Document Conventions 8](#_Toc449967590)

[1.2.1 Fonts 8](#_Toc449967591)

[1.2.2 UML Package References 9](#_Toc449967592)

[1.2.3 UML Diagrams 9](#_Toc449967593)

[1.2.3.1 Class Properties 9](#_Toc449967594)

[1.2.3.2 Diagram Icons and Arrow Types 9](#_Toc449967595)

[1.2.4 Property Table Notation 10](#_Toc449967596)

[1.2.5 Property and Class Descriptions 10](#_Toc449967597)

[1.3 Terminology 11](#_Toc449967598)

[1.4 Normative References 11](#_Toc449967599)

[2 Background Information 12](#_Toc449967600)

[2.1 Cyber Observables 12](#_Toc449967601)

[2.2 Objects 12](#_Toc449967602)

[3 Data Model 13](#_Toc449967603)

[3.1 NetworkPacketObjectType Class 13](#_Toc449967604)

[3.2 LinkLayerType Class 15](#_Toc449967605)

[3.2.1 LogicalProtocolType Class 15](#_Toc449967606)

[3.2.1.1 ARPType Class 16](#_Toc449967607)

[3.2.1.1.1 IANAHardwareType Data Type 17](#_Toc449967608)

[3.2.1.1.2 IANAEtherType Data Type 17](#_Toc449967609)

[3.2.1.1.3 ARPOpType Data Type 18](#_Toc449967610)

[3.2.1.2 NDPType Class 18](#_Toc449967611)

[3.2.1.2.1 RouterSolicitationType Class 20](#_Toc449967612)

[3.2.1.2.2 RouterSolicitationOptionsType Class 20](#_Toc449967613)

[3.2.1.2.3 RouterAdvertisementType Class 20](#_Toc449967614)

[3.2.1.2.4 RouterAdvertisementOptionsType Class 22](#_Toc449967615)

[3.2.1.2.5 NDPLinkAddrType Class 22](#_Toc449967616)

[3.2.1.2.6 NDPMTUType Class 23](#_Toc449967617)

[3.2.1.2.7 NDPPrefixInfoType Class 23](#_Toc449967618)

[3.2.1.2.8 PrefixType Class 24](#_Toc449967619)

[3.2.1.2.9 NeighborSolicitationType Class 25](#_Toc449967620)

[3.2.1.2.10 NeighborSolicitationOptionsType Class 25](#_Toc449967621)

[3.2.1.2.11 NeighborAdvertisementType Class 26](#_Toc449967622)

[3.2.1.2.12 NeighborOptionsType Class 26](#_Toc449967623)

[3.2.1.2.13 RedirectType Class 27](#_Toc449967624)

[3.2.1.2.14 RedirectOptionsType Class 27](#_Toc449967625)

[3.2.1.2.15 NDPRedirectedHeaderType Class 28](#_Toc449967626)

[3.2.2 PhysicalInterfaceType Class 28](#_Toc449967627)

[3.2.2.1 EthernetInterfaceType Class 29](#_Toc449967628)

[3.2.2.1.1 EthernetHeaderType Class 29](#_Toc449967629)

[3.2.2.1.2 TypeLengthType Class 30](#_Toc449967630)

[3.3 InternetLayerType Class 30](#_Toc449967631)

[3.3.1 IPVersionType Data Type 31](#_Toc449967632)

[3.3.2 IPv4PacketType Class 31](#_Toc449967633)

[3.3.2.1 IPv4HeaderType Class 32](#_Toc449967634)

[3.3.2.2 IANAAssignedIPNumbersType Data Type 34](#_Toc449967635)

[3.3.2.3 IPv4FlagsType Class 34](#_Toc449967636)

[3.3.2.4 DoNotFragmentType Data Type 35](#_Toc449967637)

[3.3.2.5 MoreFragmentsType Data Type 35](#_Toc449967638)

[3.3.2.6 IPv4OptionType Class 35](#_Toc449967639)

[3.3.2.7 IPv4CopyFlagType Data Type 36](#_Toc449967640)

[3.3.2.8 IPv4ClassType Data Type 36](#_Toc449967641)

[3.3.2.9 IPv4OptionsType Data Type 36](#_Toc449967642)

[3.3.3 IPv6PacketType Class 36](#_Toc449967643)

[3.3.3.1 IPv6HeaderType Class 38](#_Toc449967644)

[3.3.3.2 IPv6ExtHeaderType Class 39](#_Toc449967645)

[3.3.3.3 HopByHopOptionsType Class 40](#_Toc449967646)

[3.3.3.3.1 OptionDataType Class 41](#_Toc449967647)

[3.3.3.3.2 Pad1Type Class 41](#_Toc449967648)

[3.3.3.3.3 PadNType Class 42](#_Toc449967649)

[3.3.3.4 RoutingType Class 42](#_Toc449967650)

[3.3.3.5 FragmentType Class 43](#_Toc449967651)

[3.3.3.5.1 FragmentHeaderType Class 44](#_Toc449967652)

[3.3.3.5.2 MFlagType Data Type 44](#_Toc449967653)

[3.3.3.6 DestinationOptionsType Class 44](#_Toc449967654)

[3.3.3.7 AuthenticationHeaderType Class 45](#_Toc449967655)

[3.3.3.8 EncapsulatingSecurityPayloadType Class 46](#_Toc449967656)

[3.3.3.9 IPv6OptionType Class 47](#_Toc449967657)

[3.3.3.10 IPv6DoNotRecogActionType Data Type 48](#_Toc449967658)

[3.3.3.11 IPv6PacketChangeType Data Type 48](#_Toc449967659)

[3.3.4 ICMPv4PacketType Class 48](#_Toc449967660)

[3.3.4.1 ICMPv4HeaderType Class 50](#_Toc449967661)

[3.3.4.2 ICMPv4ErrorMessageType Class 50](#_Toc449967662)

[3.3.4.3 ICMPv4ErrorMessageContentType Class 52](#_Toc449967663)

[3.3.4.3.1 ICMPv4DestinationUnreachableType Class 53](#_Toc449967664)

[3.3.4.3.2 FragmentationRequiredType Class 55](#_Toc449967665)

[3.3.4.3.3 ICMPv4SourceQuenchType Class 55](#_Toc449967666)

[3.3.4.3.4 ICMPv4RedirectMessageType Class 56](#_Toc449967667)

[3.3.4.3.5 ICMPv4TimeExceededType Class 57](#_Toc449967668)

[3.3.4.4 ICMPv4InfoMessageType Class 57](#_Toc449967669)

[3.3.4.4.1 ICMPv4InfoMessageContentType Class 59](#_Toc449967670)

[3.3.4.4.2 ICMPv4EchoReplyType Class 60](#_Toc449967671)

[3.3.4.4.3 ICMPv4EchoRequestType Class 60](#_Toc449967672)

[3.3.4.4.4 ICMPv4TimestampRequestType Class 61](#_Toc449967673)

[3.3.4.4.5 ICMPv4TimestampReplyType Class 61](#_Toc449967674)

[3.3.4.4.6 ICMPv4AddressMaskRequestType Class 62](#_Toc449967675)

[3.3.4.4.7 ICMPv4AddressMaskReplyType Class 63](#_Toc449967676)

[3.3.4.5 ICMPv4TracerouteType Class 63](#_Toc449967677)

[3.3.5 ICMPv6PacketType Class 65](#_Toc449967678)

[3.3.5.1 ICMPv6HeaderType Class 67](#_Toc449967679)

[3.3.5.2 ICMPv6ErrorMessageType Class 67](#_Toc449967680)

[3.3.5.2.1 ICMPv6DestinationUnreachableType Class 69](#_Toc449967681)

[3.3.5.2.2 ICMPv6PacketTooBigType Class 70](#_Toc449967682)

[3.3.5.2.3 ICMPv6TimeExceededType Class 70](#_Toc449967683)

[3.3.5.2.4 ICMPv6ParameterProblemType Class 71](#_Toc449967684)

[3.3.5.3 ICMPv6InfoMessageType Class 72](#_Toc449967685)

[3.3.5.3.1 ICMPv6InfoMessageContentType Class 73](#_Toc449967686)

[3.3.5.3.2 ICMPv6EchoRequestType Class 73](#_Toc449967687)

[3.3.5.3.3 ICMPv6EchoReplyType Class 74](#_Toc449967688)

[3.4 TransportLayerType Class 74](#_Toc449967689)

[3.4.1 TCPType Class 75](#_Toc449967690)

[3.4.1.1 TCPHeaderType Class 76](#_Toc449967691)

[3.4.1.2 TCPFlagsType Class 77](#_Toc449967692)

[3.4.2 UDPType Class 78](#_Toc449967693)

[3.4.2.1 UDPHeaderType Class 79](#_Toc449967694)

[3.5 IANAPortNumberRegistryType Data Type 79](#_Toc449967695)

[3.6 Enumerations 79](#_Toc449967696)

[3.6.1 ARPOpTypeEnum Enumeration 79](#_Toc449967697)

[3.6.2 DoNotFragmentTypeEnum Enumeration 80](#_Toc449967698)

[3.6.3 MoreFragmentsTypeEnum Enumeration 80](#_Toc449967699)

[3.6.4 IPv4CopyFlagTypeEnum Enumeration 81](#_Toc449967700)

[3.6.5 IPv4ClassTypeEnum Enumeration 81](#_Toc449967701)

[3.6.6 IPv4OptionsTypeEnum Enumeration 82](#_Toc449967702)

[3.6.7 IPv6DoNotRecogActionTypeEnum Enumeration 83](#_Toc449967703)

[3.6.8 IPv6PacketChangeTypeEnum Enumeration 84](#_Toc449967704)

[3.6.9 IPVersionTypeEnum Enumeration 84](#_Toc449967705)

[3.6.10 IANAHardwareTypeEnum Enumeration 85](#_Toc449967706)

[3.6.11 IANAEtherTypeEnum Enumeration 85](#_Toc449967707)

[3.6.12 IANAAssignedIPNumbersTypeEnum Enumeration 86](#_Toc449967708)

[3.6.13 IANAPortNumberRegistryTypeEnum Enumeration 88](#_Toc449967709)

[3.6.14 MFlagTypeEnum Enumeration 88](#_Toc449967710)

[4 Conformance 90](#_Toc449967711)

[Appendix A. Acknowledgments 91](#_Toc449967712)

[Appendix B. Revision History 92](#_Toc449967713)

# Introduction

[All text is normative unless otherwise labeled]

The Cyber Observable Expression (CybOXTM) provides a common structure for representing cyber observables across and among the operational areas of enterprise cyber security. CybOX improves the consistency, efficiency, and interoperability of deployed tools and processes, and it increases overall situational awareness by enabling the potential for detailed automatable sharing, mapping, detection, and analysis heuristics.

This document serves as the specification for the CybOX Network Packet Object Version 2.1.1 data model, which is one of eighty-eight CybOX Object data models.

In Section **1.1,** we discuss additional specification documents, in Section **1.2,** we provide document conventions, and in Section **1.3,** we provide terminology. References are given in Section **1.4**. In Section **2**, we give background information necessary to fully understand the Network Packet Object data model. We present the Network Packet Object data model specification details in Section **3** and conformance information in Section **4**.

## CybOXTM Specification Documents

The CybOX specification consists of a formal UML model and a set of textual specification documents that explain the UML model. Specification documents have been written for each of the individual data models that compose the full CybOX UML model.

CybOX has a modular design comprising two fundamental data models and a collection of Object data models. The fundamental data models – CybOX Core and CybOX Common – provide essential CybOX structure and functionality. The CybOX Objects, defined in individual data models, are precise characterizations of particular types of observable cyber entities (e.g., HTTP session, Windows registry key, DNS query).

Use of the CybOX Core and Common data models is required; however, use of the CybOX Object data models is purely optional: users select and use only those Objects and corresponding data models that are needed. Importing the entire CybOX suite of data models is not necessary.

The [*CybOX Version 2.1.1 Part 1: Overview*](#AdditionalArtifacts) document provides a comprehensive overview of the full set of CybOX data models, which in addition to the Core, Common, and numerous Object data models, includes various extension data models and a vocabularies data model, which contains a set of default controlled vocabularies. [*CybOX Version 2.1.1 Part 1: Overview*](#AdditionalArtifacts) also summarizes the relationship of CybOX to other languages, and outlines general CybOX data model conventions.

## Document Conventions

The following conventions are used in this document.

### Fonts

The following font and font style conventions are used in the document:

* Capitalization is used for CybOX high level concepts, which are defined in [*CybOX Version 2.1.1 Part 1: Overview*](#AdditionalArtifacts).

Examples: Action, Object, Event, Property

* The Courier New font is used for writing UML objects.

Examples: ActionType, cyboxCommon:BaseObjectPropertyType

Note that all high level concepts have a corresponding UML object. For example, the Action high level concept is associated with a UML class named, ActionType.

* The ‘*italic’* font (withsingle quotes) is used for noting actual, explicit values for CybOX Language properties. The *italic* font (without quotes) is used for noting example values.

Example:  *‘HashNameVocab-1.0,’ high, medium, low*

### UML Package References

Each CybOX data model is captured in a different UML package (e.g., Core package) where the packages together compose the full CybOX UML model. To refer to a particular class of a specific package, we use the format package\_prefix:class, where package\_prefix corresponds to the appropriate UML package. The [*CybOX Version 2.1.1 Part 1: Overview*](#AdditionalArtifacts) document contains the full list of CybOX packages, along with the associated prefix notations, descriptions, and examples.

The package\_prefix for the Network Packet data model is PacketObj. Note that in this specification document, we do not explicitly specify the package prefix for any classes that originate from the Network Packet Object data model.

### UML Diagrams

This specification makes use of UML diagrams to visually depict relationships between CybOX Language constructs. Note that the diagrams have been extracted directly from the full UML model for CybOX; they have not been constructed purely for inclusion in the specification documents.  Typically, diagrams are included for the primary class of a data model, and for any other class where the visualization of its relationships between other classes would be useful.  This implies that there will be very few diagrams for classes whose only properties are either a data type or a class from the CybOX Common data model.  Other diagrams that are included correspond to classes that specialize a superclass and abstract or generalized classes that are extended by one or more subclasses.

In UML diagrams, classes are often presented with their attributes elided, to avoid clutter. The fully described class can usually be found in a related diagram. A class presented with an empty section at the bottom of the icon indicates that there are no attributes other than those that are visualized using associations.

#### Class Properties

Generally, a class property can be shown in a UML diagram as either an attribute or an association (i.e., the distinction between attributes and associations is somewhat subjective). In order to make the size of UML diagrams in the specifications manageable, we have chosen to capture most properties as attributes and to capture only higher level properties as associations, especially in the main top-level component diagrams. In particular, we will always capture properties of UML data types as attributes.

#### Diagram Icons and Arrow Types

Diagram icons are used in a UML diagram to indicate whether a shape is a class, enumeration, or a data type, and decorative icons are used to indicate whether an element is an attribute of a class or an enumeration literal. In addition, two different arrow styles indicate either a directed association relationship (regular arrowhead) or a generalization relationship (triangle-shaped arrowhead). The icons and arrow styles we use are shown and described in **Table 1‑1**.

Table 1‑1. UML diagram icons

|  |  |
| --- | --- |
| **Icon** | **Description** |
|  | This diagram icon indicates a class. If the name is in italics, it is an abstract class. |
|  | This diagram icon indicates an enumeration. |
|  | This diagram icon indicates a data type. |
|  | This decorator icon indicates an attribute of a class. The green circle means its visibility is public. If the circle is red or yellow, it means its visibility is private or protected. |
|  | This decorator icon indicates an enumeration literal. |
|  | This arrow type indicates a directed association relationship. |
|  | This arrow type indicates a generalization relationship. |

### Property Table Notation

Throughout Section **3**, tables are used to describe the properties of each data model class. Each property table consists of a column of names to identify the property, a type column to reflect the datatype of the property, a multiplicity column to reflect the allowed number of occurrences of the property, and a description column that describes the property. Package prefixes are provided for classes outside of the Network Packet Object data model (see Section **1.2.2**).

Note that if a class is a specialization of a superclass, only the properties that constitute the specialization are shown in the property table (i.e., properties of the superclass will not be shown). However, details of the superclass may be shown in the UML diagram.

### Property and Class Descriptions

Each class and property defined in CybOX is described using the format, “The X property verbY.” For example, in the specification for the CybOX Core data model, we write, “The id property specifies a globally unique identifier for the Action.” In fact, the verb “specifies” could have been replaced by any number of alternatives: “defines,” “describes,” “contains,” “references,” etc.

However, we thought that using a wide variety of verb phrases might confuse a reader of a specification document because the meaning of each verb could be interpreted slightly differently. On the other hand, we didn’t want to use a single, generic verb, such as “describes,” because although the different verb choices may or may not be meaningful from an implementation standpoint, a distinction could be useful to those interested in the modeling aspect of CybOX.

Consequently, we have preferred to use the three verbs, defined as follows, in class and property descriptions:

|  |  |
| --- | --- |
| **Verb** | **CybOX Definition** |
| captures | Used to record and preserve information without implying anything about the structure of a class or property. Often used for properties that encompass general content. This is the least precise of the three verbs. |
|  | *Examples*:  The Observable\_Source property characterizes the source of the Observable information. Examples of details captured include identifying characteristics, time-related attributes, and a list of the tools used to collect the information.  The Description property captures a textual description of the Action. |
| characterizes | Describes the distinctive nature or features of a class or property. Often used to describe classes and properties that themselves comprise one or more other properties. |
|  | *Examples*:  The Action property characterizes a cyber observable Action.  The Obfuscation\_Technique property characterizes a technique an attacker could potentially leverage to obfuscate the Observable. |
| specifies | Used to clearly and precisely identify particular instances or values associated with a property. Often used for properties that are defined by a controlled vocabulary or enumeration; typically used for properties that take on only a single value. |
|  | *Example*:  The cybox\_major\_version property specifies the major version of the CybOX language used for the set of Observables. |

## Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in **[**RFC2119**]**.

## Normative References

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”, BCP 14, RFC 2119, March 1997. <http://www.ietf.org/rfc/rfc2119.txt>.

# Background Information

In this section, we provide high level information about the Network Packet Object data model that is necessary to fully understand the specification details given in Section **3**.

## Cyber Observables

A cyber observable is a dynamic event or a stateful property that occurs, or may occur, in the operational cyber domain. Examples of stateful properties include the value of a registry key, the MD5 hash of a file, and an IP address. Examples of events include the deletion of a file, the receipt of an HTTP GET request, and the creation of a remote thread.

A cyber observable is different than a cyber indicator. A cyber observable is a statement of fact, capturing what was observed or could be observed in the cyber operational domain. Cyber indicators are cyber observable patterns, such as a registry key value associated with a known bad actor or a spoofed email address used on a particular date.

## Objects

Cyber observable objects (Files, IP Addresses, etc) in CybOX are characterized with a combination of two levels of data models.

The first level is the Object data model which specifies a base set of properties universal to all types of Objects and enables them to integrate with the overall cyber observable framework specified in the CybOX Core data model.

The second level are the object property models which specify the properties of a particular type of Object via individual data models each focused on a particular cyber entity, such as a Windows registry key, or an Email Message. Accordingly, each release of the CybOX language includes a particular set of Objects that are part of the release. The data model for each of these Objects is defined by its own specification that describes the context-specific classes and properties that compose the Object.

Any specific instance of an Object is represented utilizing the particular object properties data model within the general Object data model.

# Data Model

## NetworkPacketObjectType Class

The NetworkPacketObjectType class is based on the TCP/IP model/Internet protocol suite. In the TCP/IP stack, a "packet" is generally defined as IP header plus payload, but we also include the Link Layer, which defines the physical network interfaces and routing protocols and the Transport Layer, which defines provide host-to-host communication services for applications. Protocol fields are provided but requirements are not enforced/captured.

See <https://tools.ietf.org/html/rfc1122> for more information.

In CybOX 2.1.1, the properties of the NetworkPacketObjectType class are mutually exclusive, i.e., only one property can be populated. In future releases, this restriction will probably be removed.

The UML diagram corresponding to the NetworkPacketObjectType class is shown in **Figure 3‑1**.

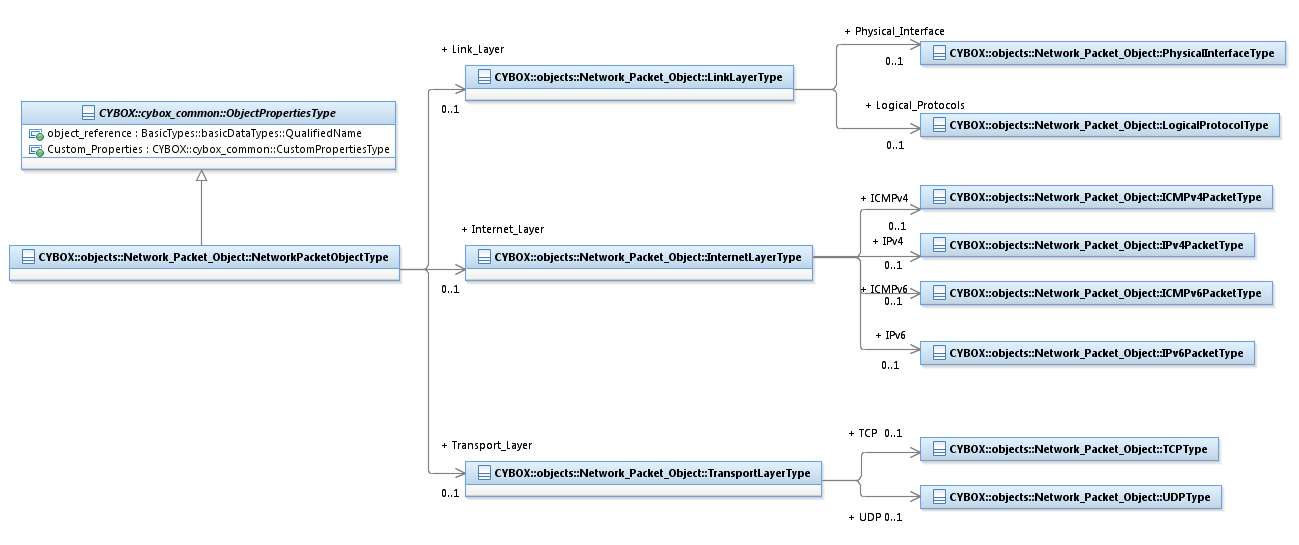


Figure 3‑1. UML diagram of the NetworkPacketObjectType class

The property table of the NetworkPacketObjectType class is given in **Table 3‑1**.

Table 3‑1. Properties of the NetworkPacketObjectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Link\_Layer** | LinkLayerType | 0..1 | The Link\_Layer property is the lowest layer of the TCP/IP network stack and is comprised of physical and logical protocols that operate between adjacent nodes of a network segment or a WAN connection.  Only one of the properties of NetworkPacketObjectType can be populated. |
| **Internet\_Layer** | InternetLayerType | 0..1 | The Internet\_Layer property characterizes information about the network layer of this packet, as defined in the 7-layer OSI model. See <https://en.wikipedia.org/wiki/OSI_model> for more details.  Only one of the properties of NetworkPacketObjectType can be populated. |
| **Transport\_Layer** | TransportLayerType | 0..1 | The Transport\_Layer property characterizes information about the transport layer of this packet as defined in the 7-layer OSI model. See <https://en.wikipedia.org/wiki/OSI_model> for more details.  Only one of the properties of NetworkPacketObjectType can be populated. |

## LinkLayerType Class

The LinkLayerType class characterizes the link layer protocol, which is a hardware interface protocol, such as Ethernet, or a logical link routing protocol, such as ARP.

In CybOX 2.1.1, the properties of the LinkLayerType class are mutually exclusive, i.e., only one property can be populated. In future releases, this restriction will probably be removed.

The property table of the LinkLayerType class is given in **Table 3‑2**.

Table 3‑2. Properties of the LinkLayerType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Physical\_Interface** | PhysicalInterfaceType | 0..1 | The Physical\_Interface property characterizes one hardware interface of a link layer connection.  Only one of the properties of LinkLayerType can be populated. |
| **Logical\_Protocols** | LogicalProtocolType | 0..1 | The Logical\_Protocols property characterizes the logical protocol of a link layer connection. One example of a logical protocol is ARP.  Only one of the properties of LinkLayerType can be populated. |

### LogicalProtocolType Class

The LogicalProtocolType class characterizes the logical protocol of a link layer connection. One example of a logical protocol is ARP.

In CybOX 2.1.1, the properties of the LogicalProtocolType class are mutually exclusive, i.e., only one property can be populated. This restriction is based on the fact that the ARPType class and the NDPType class can be modelled as subclasses of the LogicalProtocolType class. In future releases, it will probably be modelled that way.

The property table of the LogicalProtocolType class is given in **Table 3‑3**.

Table 3‑3. Properties of the LogicalProtocolType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **ARP\_RARP** | ARPType | 0..1 | The ARP\_RARP property specifies either the ARP or the RARP logical protocol. ARP is a logical protocol used for resolution of network layer addresses (e.g., IP addresses) into link layer addresses (e.g., MAC addresses). RARP is a logical protocol used by a host computer to request its network layer address when it has its link layer address.  Only one of the properties of LogicalProtocolType can be populated. |
| **NDP** | NDPType | 0..1 | The NDP property specifies the Neighbor Discovery Protocol, which is used with IPv6 to determine the link-layer addresses for neighbors. It corresponds to a combination of IPv4 protocols: ARP, ICMP Router Discovery, and ICMP Redirect.  Only one of the properties of LogicalProtocolType can be populated. |

#### ARPType Class

The ARPType class characterizes the Address Resolution Protocol, which is a request and reply protocol that runs encapsulated by the line protocol. It is communicated within the boundaries of a single network, and never routed across inter-network nodes. This property places ARP into the Link Layer. See <http://www.comptechdoc.org/independent/networking/guide/netarp.html> for more information.

The property table of the ARPType class is given in **Table 3‑4**.

Table 3‑4. Properties of the ARPType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Hardware\_Addr\_Type** | IANAHardwareType | 0..1 | The Hardware\_Addr\_Type property characterizes the type of hardware address specified in an ARP message. |
| **Proto\_Addr\_Type** | IANAEtherType | 0..1 | The Proto\_Addr\_Type property characterizes the type of protocol address being mapped. For IPv4 addresses, the value is 0x0800. |
| **Hardware\_Addr\_Size** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Hardware\_Addr\_Size property represents the byte size of the hardware address. For Ethernet or other IEEE 802 MAC addresses, the value is 6. |
| **Proto\_Addr\_Size** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Proto\_Addr\_Size property represents the byte size of the protocol address. For IPv4 addresses, the value is 4. |
| **Op\_Type** | ARPOpType | 0..1 | The Op\_Type property characterizes the type of operation. 1 for an ARP request, 2 for a ARP reply, 3 for a RARP request and 4 for a RARP reply. |
| **Sender\_Hardware\_Addr** | AddressObj:  AddressObjectType | 0..1 | The Sender\_Hardware\_Addr property characterizes the sender's hardware address (e.g., MAC address). |
| **Sender\_Protocol\_Addr** | AddressObj:  AddressObjectType | 0..1 | The Sender\_Protocol\_Addr property characterizes the sender's IP address. |
| **Recip\_Hardware\_Addr** | AddressObj:  AddressObjectType | 0..1 | The Recip\_Hardware\_Addr property characterizes the recipients' hardware address (e.g., MAC address). |
| **Recip\_Protocol\_Addr** | AddressObj:  AddressObjectType | 0..1 | The Recip\_Protocol\_Addr property characterizes the recipient's IP address. |

##### IANAHardwareType Data Type

The IANAHardwareType data type specifies the type of hardware. Its core value SHOULD be a literal found in the IANAHardwareTypeEnum enumeration. Its base type is the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

##### IANAEtherType Data Type

The IANAEtherType data type specifies the protocol type. Its core value SHOULD be a literal found in the IANAEtherTypeEnum enumeration. Its base type is the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

##### ARPOpType Data Type

The ARPOpType data type specifies the type of ARP operations. Its core value SHOULD be a literal found in the ARPOpTypeEnum enumeration. Its base type is the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

#### NDPType Class

The NDPType class characterizes a Neighbor Discover Protocol (NDP) IPv6 packet. There are five different types of NDP packets. See <http://tools.ietf.org/html/rfc4861> for more information.

In CybOX 2.1.1, all of the properties of the NDPType class, except ICMPv6\_Header, are mutually exclusive, i.e., only one property can be populated. This restriction is based on the fact that the classes RouterSolicitationType, RouterAdvertisementType, NeighborSolicitationType, NeighborAdvertisementType and RedirectType can be modelled as subclasses of the NDPType class. In future releases, it will probably be modelled that way.

The property table of the NDPType class is given in **Table 3‑5**.

Table 3‑5. Properties of the NDPType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **ICMPv6\_Header** | ICMPv6HeaderType | 0..1 | The ICMPv6\_Header property characterizes an ICMPv6 header. |
| **Router\_Solicitation** | RouterSolicitationType | 0..1 | The Router\_Solicitation property is used to specify a RouterSolicitationType form of the NDPType class. Hosts send Router Solicitations in order to prompt routers to generate Router Advertisements quickly (type=133; code=0).  Only one of the Router\_Solicitation, Router\_Advertisement, Neighbor\_Solicitation, Neighbor\_Advertisement and Redirect properties can be populated. |
| **Router\_Advertisement** | RouterAdvertisementType | 0..1 | The Router\_Advertisement property is used to specify a RouterAdvertisementType form of the NDPType class. Routers send out Router Advertisement messages periodically, or in response to Router Solicitations (type=134; code=0).  Only one of the Router\_Solicitation, Router\_Advertisement, Neighbor\_Solicitation, Neighbor\_Advertisement and Redirect properties can be populated. |
| **Neighbor\_Solicitation** | NeighborSolicitationType | 0..1 | The Neighbor\_Solicitation property is used to specify a NeighborSolicitationType form of the NDPType class. Nodes send Neighbor Solicitations to request the link-layer address of a target node while also providing their own link-layer address to the target. Neighbor Solicitations are multicast when the node needs to resolve an address and unicast when the node seeks to verify the reachability of a neighbor (type=135; code=0).  Only one of the Router\_Solicitation, Router\_Advertisement, Neighbor\_Solicitation, Neighbor\_Advertisement and Redirect properties can be populated. |
| **Neighbor\_Advertisement** | NeighborAdvertisementType | 0..1 | The Neighbor\_Advertisement property is used to specify a NeighborAdvertisementType form of the NDPType class. A node sends Neighbor Advertisements in response to Neighbor Solicitations and sends unsolicited Neighbor Advertisements in order to (unreliably) propagate new information quickly (type=136; code=0).  Only one of the Router\_Solicitation, Router\_Advertisement, Neighbor\_Solicitation, Neighbor\_Advertisement and Redirect properties can be populated. |
| **Redirect** | RedirectType | 0..1 | The Redirect property is used to specify a RedirectType form of the NDPType class. Routers send Redirect packets to inform a host of a better first-hop node on the path to a destination. Hosts can be redirected to a better first-hop router but can also be informed by a redirect that the destination is in fact a neighbor. The latter is accomplished by setting the ICMP Target Address equal to the ICMP Destination Address (type=137; code=0).  Only one of the Router\_Solicitation, Router\_Advertisement, Neighbor\_Solicitation, Neighbor\_Advertisement and Redirect properties can be populated. |

##### RouterSolicitationType Class

The RouterSolicitationType class specifies a NDP packet which hosts send in order to prompt routers to generate Router Advertisements quickly (class=133; code=0).

The property table of the RouterSolicitationType class is given in **Table 3‑6**.

Table 3‑6. Properties of the RouterSolicitationType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Options** | RouterSolicitationOptionsType | 0..\* | The Options property - Router Solicitation messages include zero or more options, some of which may appear multiple times in the same message. |

##### RouterSolicitationOptionsType Class

The RouterSolicitationOptionsType class specifies zero or more options for a Router Solicitation packet, some of which may appear multiple times in the same message.

The property table of the RouterSolicitationOptionsType class is given in **Table 3‑7**.

Table 3‑7. Properties of the RouterSolicitationOptionsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Src\_Link\_Addr** | NDPLinkAddrType | 0..1 | The Src\_Link\_Addr property characterizes the Source Link-Layer Address option. |

##### RouterAdvertisementType Class

The RouterAdvertisementType class specifies a NDP packet which routers send periodically, or in response to Router Solicitations (class=134; code=0).

The property table of the RouterAdvertisementType class is given in **Table 3‑8**.

Table 3‑8. Properties of the RouterAdvertisementType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **managed\_address\_**  **config\_flag** | basicDataTypes:Boolean | 0..1 | The managed\_address\_config\_flag property specifies the 1-bit "Managed address configuration" flag. When set, it indicates that addresses are available via Dynamic Host Configuration Protocol. If the M flag is set, the O flag is redundant and can be ignored because DHCPv6 will return all available configuration information. |
| **other\_config\_flag** | basicDataTypes:Boolean | 0..1 | The other\_config\_flag property specifies the 1-bit "Other configuration" flag. When set, it indicates that other configuration information is available via DHCPv6. Examples of such information are DNS-related information or information on other servers within the network. |
| **Cur\_Hop\_Limit** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Cur\_Hop\_Limit property is an 8-bit unsigned integer, which specifies the current hop limit. The default value that should be placed in the Hop Count property of the IP header for outgoing IP packets. A value of zero means unspecified (by this router). |
| **Router\_Lifetime** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Router\_Lifetime property is a 16-bit unsigned integer which specifies the lifetime associated with the default router, in units of seconds. The property can contain values up to 65535 and receivers should handle any value, while the sending rules in Section 6 limit the lifetime to 9000 seconds. |
| **Reachable\_Time** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Reachable\_Time property is a 32-bit unsigned integer which specifies the time, in milliseconds, between retransmitted Neighbor Solicitation messages. Used by address resolution and the Neighbor Unreachability Detection algorithm. A value of zero means unspecified (by this router). |
| **Retrans\_Timer** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Retrans\_Timer property is a 32-bit unsigned integer which specifies the time, in milliseconds, between retransmitted Neighbor Solicitation messages. Used by address resolution and the Neighbor Unreachability Detection algorithm. A value of zero means unspecified (by this router). |
| **Options** | RouterAdvertisementOptionsType | 0..1 | The Options property specifies zero or more options, some of which may appear multiple times in the same message. |

##### RouterAdvertisementOptionsType Class

The RouterAdvertisementOptionsType class specifies zero or more options of a Router Advertisement packet, some of which may appear multiple times in the same message.

The property table of the RouterAdvertisementOptionsType class is given in **Table 3‑9**.

Table 3‑9. Properties of the RouterAdvertisementOptionsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Src\_Link\_Addr** | NDPLinkAddrType | 0..1 | The Src\_Link\_Addr property characterizes the Source Link-Layer Address option. |
| **MTU** | NDPMTUType | 0..1 | The MTU property is a 32-bit unsigned integer, which specifies the recommended MTU for the link. |
| **Prefix\_Info** | NDPPrefixInfoType | 0..1 | The Prefix\_Info property characterizes the Prefix Information for the Router Advertisement Options. |

##### NDPLinkAddrType Class

The NDPLinkAddrType class characterizes the Link-Layer Address option.

The property table of the NDPLinkAddrType class is given in **Table 3‑10**.

Table 3‑10. Properties of the NDPLinkAddrType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Length** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Length property of the option (including the type and length properties) in units of 8 octets. |
| **Link\_Layer\_MAC\_Addr** | AddressObj:  AddressObjectType | 0..1 | The Link\_Layer\_MAC\_Addr property specifies a variable length link-layer address. The content and format of this property (including byte and bit ordering) is expected to be specified in specific documents that describe how IPv6 operates over different link layers. |

##### NDPMTUType Class

The NDPMTUType class specifies the MTU option which is used in Router Advertisement messages to ensure that all nodes on a link use the same MTU value in those cases where the link MTU is not well known (class=5).

The property table of the NDPMTUType class is given in **Table 3‑11**.

Table 3‑11. Properties of the NDPMTUType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Length** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Length property specifies the length of the MTU option type: length=1. |
| **MTU** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The MTU property is a 32-bit unsigned integer that specifies the recommended MTU for the link. |

##### NDPPrefixInfoType Class

The NDPPrefixInfoType class characterizes the prefix information for Router Advertisement Options. It provides hosts with on-link prefixes and prefixes for Address Autoconfiguration (class=3). See <http://tools.ietf.org/html/rfc4861> for more information.

The property table of the NDPPrefixInfoType class is given in **Table 3‑12**.

Table 3‑12. Properties of the NDPPrefixInfoType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **link\_flag** | basicDataTypes:Boolean | 0..1 | The link\_flag property specifies the 1-bit on-link flag. When set, it indicates that this prefix can be used for on-link determination. When not set the advertisement makes no statement about on-link or off-link properties of the prefix. |
| **addr\_config\_flag** | basicDataTypes:Boolean | 0..1 | The addr\_config\_flag property specifies the 1-bit autonomous address-configuration flag. When set, it indicates that this prefix can be used for stateless address configuration. |
| **Length** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Length property characterizes the length of the option (the number of valid leading bits in the prefix), and is represented as a 32-bit integer. |
| **Prefix\_Length** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Prefix\_Length property contains an 8-bit unsigned integer specifying the number of leading bits in the Prefix that are valid. The value ranges from 0 to 128. The prefix length property provides necessary information for on-link determination (when combined with the L flag in the prefix information option). |
| **Valid\_Lifetime** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Valid\_Lifetime property contains a 32-bit unsigned integer specifying the length of time in seconds (relative to the time the packet is sent) that the prefix is valid for the purpose of on-link determination. A value of all one bits (0xffffffff) represents infinity. |
| **Preferred\_Lifetime** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Preferred\_Lifetime property contains a 32-bit unsigned integer, specifying the length of time in seconds (relative to the time the packet is sent) that addresses generated from the prefix via stateless address autoconfiguration remain preferred. |
| **Prefix** | PrefixType | 0..1 | The Prefix property specifies an IP address or a prefix of an IP address for IPv6. |

##### PrefixType Class

The PrefixType class characterizes an IP address or a prefix of an IP address for IPv6.

The property table of the PrefixType class is given in **Table 3‑13**.

Table 3‑13. Properties of the PrefixType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **IPv6\_Addr** | AddressObj:AddressObjectType | 0..1 | The IPv6\_Addr property specifies the IPv6 address. |
| **IP\_Addr\_Prefix** | AddressObj:AddressObjectType | 0..1 | The IP\_Addr\_Prefix property specifies the initial bits of an IPv6 address (these are identical for all hosts in a network) from the network's prefix. See <http://ipv6.com/articles/general/IPv6-Addressing.htm> for more information. |

##### NeighborSolicitationType Class

The NeighborSolicitationType class specifies a NDP packet which nodes send to request the link-layer address of a target node while also providing their own link-layer address to the target. Neighbor Solicitations are multicast when the node needs to resolve an address and unicast when the node seeks to verify the reachability of a neighbor (class=135; code=0).

The property table of the NeighborSolicitationType class is given in **Table 3‑14**.

Table 3‑14. Properties of the NeighborSolicitationType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Target\_IPv6\_Addr** | AddressObj:AddressObjectType | 0..1 | The Target\_IPv6\_Addr property specifies the IP address of the target of the solicitation. |
| **Options** | NeighborSolicitationOptionsType | 0..1 | The Options property specifies zero or more options, some of which may appear multiple times in the same message. |

##### NeighborSolicitationOptionsType Class

The NeighborSolicitationOptionsType class specifies zero or more options of a Neighbor Solicitation packet, some of which may appear multiple times in the same message.

The property table of the NeighborSolicitationOptionsType class is given in **Table 3‑15**.

Table 3‑15. Properties of the NeighborSolicitationOptionsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Src\_Link\_Addr** | NDPLinkAddrType | 0..1 | The Src\_Link\_Addr property characterizes the Source Link-Layer Address option. |

##### NeighborAdvertisementType Class

The NeighborAdvertisementType class specifies a NDP packet which a node sends in response to Neighbor Solicitations and sends unsolicited Neighbor Advertisements in order to (unreliably) propagate new information quickly (class=136; code=0).

The property table of the NeighborAdvertisementType class is given in **Table 3‑16**.

Table 3‑16. Properties of the NeighborAdvertisementType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **router\_flag** | basicDataTypes:  Boolean | 0..1 | The router\_flag property specifies the router flag (R-bit). When set, the R-bit indicates that the sender is a router. The R-bit is used by Neighbor Unreachability Detection to detect a router that changes to a host. |
| **solicited\_flag** | basicDataTypes:  Boolean | 0..1 | The solicited\_flag property specifies the solicited flag (S-bit). When set, the S-bit indicates that the advertisement was sent in response to a Neighbor Solicitation from the Destination address. The S-bit is used as a reachability confirmation for Neighbor Unreachability Detection. |
| **override\_flag** | basicDataTypes:  Boolean | 0..1 | The override\_flag property specifies the override flag (O-bit). When set, the O-bit indicates that the advertisement should override an existing cache entry and update the cached link-layer address. |
| **Target\_IPv6\_Addr** | AddressObj:  AddressObjectType | 0..1 | The Target\_IPv6\_Addr property specifies the IP address of the target of the advertisement. |
| **Options** | NeighborOptionsType | 0..1 | The Options property specifies zero or more options, some of which may appear multiple times in the same message. |

##### NeighborOptionsType Class

The NeighborOptionsType class specifies zero or more options of a Neighbor Advertisement packet, some of which may appear multiple times in the same message.

The property table of the NeighborOptionsType class is given in **Table 3‑17**.

Table 3‑17. Properties of the NeighborOptionsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Target\_Link\_Addr** | NDPLinkAddrType | 0..1 | The Target\_Link\_Addr property characterizes the Target Link-Layer Address option. |

##### RedirectType Class

The RedirectType class specifies a NDP packet which Routers send to inform a host of a better first-hop node on the path to a destination. Hosts can be redirected to a better first-hop router but can also be informed by a redirect that the destination is in fact a neighbor. The latter is accomplished by setting the ICMP Target Address equal to the ICMP Destination Address (class=137; code=0).

The property table of the RedirectType class is given in **Table 3‑18**.

Table 3‑18. Properties of the RedirectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Target\_IPv6\_Addr** | AddressObj:  AddressObjectType | 0..1 | The Target\_IPv6\_Addr property specifies an IP address that is a better first hop to use for the ICMP Destination Address. |
| **Dest\_IPv6\_Addr** | AddressObj:  AddressObjectType | 0..1 | The Dest\_IPv6\_Addr property specifies the IP address of the destination that is redirected to the target. |
| **Options** | RedirectOptionsType | 0..1 | The Options property specifies zero or more options, some of which may appear multiple times in the same message. |

##### RedirectOptionsType Class

The RedirectOptionsType class specifies zero or more options of a Redirect packet, some of which may appear multiple times in the same message.

The property table of the RedirectOptionsType class is given in **Table 3‑19**.

Table 3‑19. Properties of the RedirectOptionsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Target\_Link\_Addr** | NDPLinkAddrType | 0..1 | The Target\_Link\_Addr property specifies the link-layer address for the target. |
| **Redirected\_Header** | NDPRedirectedHeaderType | 0..1 | The Redirected\_Header property specifies as much of the IP packet as possible that triggered the sending of the Redirect message without making the redirect packet exceed the minimum MTU specified in the IPv6 protocol. |

##### NDPRedirectedHeaderType Class

The NDPRedirectedHeaderType class is used in redirect packets and contains all or part of the packet that is being redirected (class=4).

The property table of the NDPRedirectedHeaderType class is given in **Table 3‑20**.

Table 3‑20. Properties of the NDPRedirectedHeaderType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Length** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Length property specifies the length of the option (including the type and length properties) in units of 8 octets. |
| **IPHeader\_And\_Data** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The IPHeader\_And\_Data property specifies as much of the IP packet that triggered the sending of the redirect without making redirect packet larger than MTU. |

### PhysicalInterfaceType Class

The PhysicalInterfaceType class specifies the interface to the physical network. Multiple interface classes exist, however, only the most common (Ethernet) is defined. Others will be added later as needed.

The property table of the PhysicalInterfaceType class is given in **Table 3‑21**.

Table 3‑21. Properties of the PhysicalInterfaceType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Ethernet** | EthernetInterfaceType | 0..1 | The Ethernet property specifies the Ethernet interface that sends network packets from the sending host to one or more receiving hosts. See <http://www.ieee802.org/3/> and <http://wiki.wireshark.org/Ethernet> for more information. |

#### EthernetInterfaceType Class

The EthernetInterfaceType class characterizes the Ethernet interface, which is used to send network packets from the sending host to one or more receiving hosts.

See <http://www.ieee802.org/3/> and <http://wiki.wireshark.org/Ethernet> for more information.

The property table of the EthernetInterfaceType class is given in **Table 3‑22**.

Table 3‑22. Properties of the EthernetInterfaceType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Ethernet\_Header** | EthernetHeaderType | 0..1 | The Ethernet\_Header property includes information such as source MAC address, destination MAC address, and more. |

##### EthernetHeaderType Class

The EthernetHeaderType class characterizes the Ethernet header and includes information such as source MAC address, destination MAC address, and more.

The property table of the EthernetHeaderType class is given in **Table 3‑23**.

Table 3‑23. Properties of the EthernetHeaderType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Destination\_MAC\_Addr** | AddressObj:  AddressObjectType | 0..1 | The Destination\_MAC\_Addr property characterizes the destination MAC Address of the Ethernet frame. |
| **Source\_MAC\_Addr** | AddressObj:  AddressObjectType | 0..1 | The Source\_MAC\_Addr property characterizes the source MAC Address of the Ethernet frame. |
| **Type\_Or\_Length** | TypeLengthType | 0..1 | The Type\_Or\_Length property characterizes either the length of the Ethernet frame or the protocol type of the network layer. |
| **Checksum** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Checksum property characterizes the Frame Check sequence of an Ethernet frame. |

##### TypeLengthType Class

The TypeLengthType class can specify either the length or the protocol type. If the value is 0-1500, then a length is being specified. Otherwise, it specifies the protocol class of the Internet layer.

The property table of the TypeLengthType class is given in **Table 3‑24**.

Table 3‑24. Properties of the TypeLengthType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Length** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Length property characterizes the length of the Ethernet frame. |
| **Internet\_Layer\_Type** | IANAEtherType | 0..1 | The Internet\_Layer\_Type property consists of two octets in an Ethernet frame, and specifies the protocol encapsulated in the payload of frame. |

## InternetLayerType Class

The InternetLayerType class specifies the group of methods, protocols, and specifications that are used to transport packets from the originating host across network boundaries. Only protocols most commonly used are currently defined: IPv4, ICMPv4, IPv6 and ICMPv6. Other protocols will be added as needed. See <http://en.wikipedia.org/wiki/Internet_layer> for more information.

In CybOX 2.1.1, all of the properties of the InternetLayerType class are mutually exclusive, i.e., only one property can be populated. This restriction is based on the fact that the classes IPv4PacketType, ICMPv4PacketType, IPv6PacketType and ICMPv6PacketType can be modelled as subclasses of the InternetLayerType class. In future releases, it will probably be modelled that way.

The property table of the InternetLayerType class is given in **Table 3‑25**.

Table 3‑25. Properties of the InternetLayerType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **IPv4** | IPv4PacketType | 0..1 | The IPv4 property specifies the Internet Protocol version 4 (IPv4) packet. IPv4 is a connectionless protocol for use on packet-switched link layer networks (e.g., Ethernet).  Only one of the properties of InternetLayerType can be populated. |
| **ICMPv4** | ICMPv4PacketType | 0..1 | The ICMPv4 property specifies an ICMP packet (v4) which is chiefly used in the operating systems of networked computers to send error messages indicating, for example, that a requested service is not available or that a host or router could not be reached. See <http://en.wikipedia.org/wiki/Internet_Control_Message_Protocol> and <http://www.networksorcery.com/enp/protocol/icmp.htm> for more information.  Only one of the properties of InternetLayerType can be populated. |
| **IPv6** | IPv6PacketType | 0..1 | The IPv6 property specifies the Internet Protocol version 6 (IPv6) packets. IPv6 is a connectionless protocol for use on packet-switched link layer networks which is intended to succeed IPv4.  Only one of the properties of InternetLayerType can be populated. |
| **ICMPv6** | ICMPv6PacketType | 0..1 | The ICMPv6 property specifies an ICMP packet (v6). ICMPv6 performs error reporting and diagnostic functions.  Only one of the properties of InternetLayerType can be populated. |

### IPVersionType Data Type

The IPVersionType data type specifies the version of the IP protocol is being used. Its core value SHOULD be a literal found in the IPVersionTypeEnum enumeration. Its base type is the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

### IPv4PacketType Class

The IPv4PacketType class is used to characterize a packet using the Internet Protocol version 4 (IPv4), which is a connectionless protocol for use on packet-switched link layer networks (e.g., Ethernet). See <http://tools.ietf.org/html/rfc791> and <http://en.wikipedia.org/wiki/IPv4> for more information.

The UML diagram corresponding to the IPv4PacketType class is shown in **Figure 3‑2**.

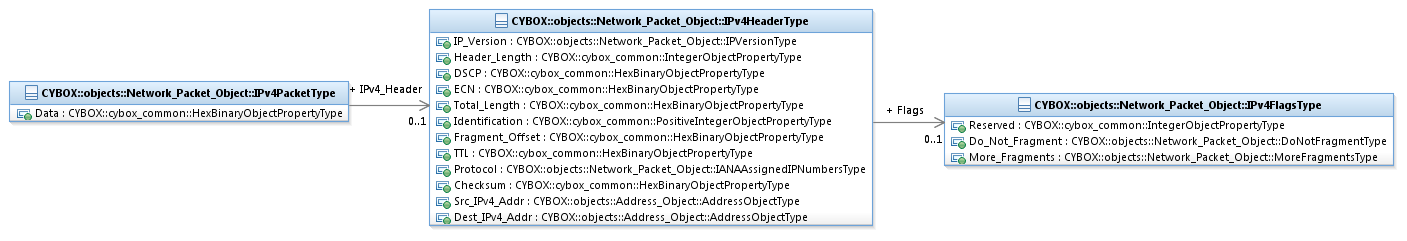


Figure 3‑2. UML diagram for the IPv4PacketType class

The property table of the IPv4PacketType class is given in **Table 3‑26**.

Table 3‑26. Properties of the IPv4PacketType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **IPv4\_Header** | IPv4HeaderType | 0..1 | The IPv4\_Header property specifies the IPv4 header, which provides addressing, and internet modules use properties in the header to fragment and reassemble internet datagrams when necessary for transmission through small packet networks. |
| **Data** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Data property, which contains the data portion of an IP packet, is interpreted based on the value of the Protocol header property. Actual property values will probably be specified in the elements of the different network layers, but we provide a property here to capture any data as necessary. |

#### IPv4HeaderType Class

The IPv4HeaderType class characterizes the IPv4 header, which provides addressing, and internet modules use fields in the header to fragment and reassemble internet datagrams when necessary for transmission through small packet networks. See <http://tools.ietf.org/html/rfc791> and <http://en.wikipedia.org/wiki/IPv4> for more information.

The property table of the IPv4HeaderType class is given in **Table 3‑27**.

Table 3‑27. Properties of the IPv4HeaderType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **IP\_Version** | IPVersionType | 0..1 | The IP\_Version property indicates the format of the internet header. For this class, the version is specified using the enumeration literal IPv4(4). |
| **Header\_Length** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Header\_Length property specifies the length of IP packet header in 32 bit words. The minimum value is 5. |
| **DSCP** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The DSCP property specifies the Differentiated Services Code Point (DSCP) as defined by <http://tools.ietf.org/html/rfc2474>. New technologies are emerging that require real-time data streaming and therefore make use of the DSCP field. An example is Voice over IP (VoIP), which is used for interactive data voice exchange. |
| **ECN** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The ECN property specifies the Explicit Congestion Notification as defined in [http://tools.ietf.org/html/rfc3168](http://tools.ietf.org/html/rfc3168%20). The ECN allows end-to-end notification of network congestion without dropping packets. ECN is an optional feature that is only used when both endpoints support it and are willing to use it. It is only effective when supported by the underlying network. |
| **Total\_Length** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Total\_Length property is a 16-bit property which specifies the entire datagram size, including header and data, in bytes. |
| **Identification** | cyboxCommon:  PositiveIntegerObjectPropertyType | 0..1 | The Identification property is used to uniquely identify fragments of an original IP datagram. |
| **Flags** | IPv4FlagsType | 0..1 | The Flags property is used to specify the three-bit property used to control or identify fragments. |
| **Fragment\_Offset** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Fragment\_Offset property is 13 bits long and specifies the offset of a particular fragment relative to the beginning of the original unfragmented IP datagram. |
| **TTL** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The TTL property that specifies an 8-bit property that helps prevent datagrams from persisting on an internet (it limits a datagram's lifetime). |
| **Protocol** | IANAAssignedIPNumbersType | 0..1 | The Protocol property specifies the protocol used in the data portion of the IP datagram. The type of this property is an enumerated list of IP protocol numbers as maintained by the Internet Assigned Numbers Authority (IANA). |
| **Checksum** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Checksum property specifies a 16-bit checksum used for error-checking of the header. |
| **Src\_IPv4\_Addr** | AddressObj:AddressObjectType | 0..1 | The Src\_IPv4\_Addr property specifies the IPv4 address of the sender of the packet. |
| **Dest\_IPv4\_Addr** | AddressObj:AddressObjectType | 0..1 | The Desc\_IPv4\_Addr property specifies the IPv4 address of the receiver of the packet. |
| **Option** | IPv4OptionType | 0..\* | The Option property is variable in length with zero or more options. It is not often used. |

#### IANAAssignedIPNumbersType Data Type

The IANAAssignedIPNumbersType data type specifies the Internet Protocol numbers. Its core value SHOULD be a literal found in the IANAAssignedIPNumbersTypeEnum enumeration. Its base type is the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

#### IPv4FlagsType Class

The IPv4FlagsType class specifies the flags that control or identify fragments in an IP packet. It is a three-bit field, each of the three bits are defined by a property with an enumeration value that indicates the meaning of whether or not the bit is set.

The property table of the IPv4FlagsType class is given in **Table 3‑28**.

Table 3‑28. Properties of the IPv4FlagsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Reserved** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Reserved property corresponds to bit 0: This bit value (0) is reserved and must be zero. |
| **Do\_Not\_Fragment** | DoNotFragmentType | 0..1 | The Do\_Not\_Fragment property corresponds to bit 1: This is the "don't fragment" bit. |
| **More\_Fragments** | MoreFragmentsType | 0..1 | The More\_Fragments property corresponds to bit 2: This is the "more fragments" bit. |

#### DoNotFragmentType Data Type

The DoNotFragmentType data type specifies the fragmenting option. Its core value SHOULD be a literal found in the DoNotFragmentTypeEnum enumeration. Its base type is the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

#### MoreFragmentsType Data Type

The MoreFragmentsType data type specifies whether there are more fragments. Its core value SHOULD be a literal found in the MoreFragmentsTypeEnum enumeration. Its base type is the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

#### IPv4OptionType Class

The IPv4OptionType class specifies the zero or more options of the IPv4 packet.

The property table of the IPv4OptionType class is given in **Table 3‑29**.

Table 3‑29. Properties of the IPv4OptionType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Copy\_Flag** | IPv4CopyFlagType | 0..1 | The Copy\_Flag property specifies the 1 bit which indicates that this option is copied into all fragments on fragmentation. |
| **Class** | IPv4ClassType | 0..1 | The Class property specifies the class type of the packet and corresponds to the 2 bit field, where 0 = control; 1 = reserved for future use; 2 = debugging and measurement; 3 = reserved for future use. |
| **Option** | IPv4OptionsType | 0..1 | The Option property specifies the optional header properties identified by an option type. |

#### IPv4CopyFlagType Data Type

The IPv4CopyFlagType data type specifies the value of IPv4 copy flag. Its core value SHOULD be a literal found in the IPv4CopyFlagTypeEnum enumeration. Its base type is the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

#### IPv4ClassType Data Type

The IPv4ClassType data type specifies the IPv4 class. Its core value SHOULD be a literal found in the IPv4ClassTypeEnum enumeration. Its base type is the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

#### IPv4OptionsType Data Type

The IPv4OptionsType data type specifies the IPv4 options. Its core value SHOULD be a literal found in the IPv4OptionsTypeEnum enumeration. Its base type is the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

### IPv6PacketType Class

The IPv4PacketType class is used to characterize a packet using the Internet Protocol version 6 (IPv6) which is intended to succeed IPv4. Like IPv4, it is a connectionless protocol for use on packet-switched link layer networks. See <http://tools.ietf.org/html/rfc3513>, <http://tools.ietf.org/html/rfc2460> and <http://en.wikipedia.org/wiki/IPv6> for more information. The UML diagram corresponding to the IPv6PacketType class is shown in **Figure 3‑3**.

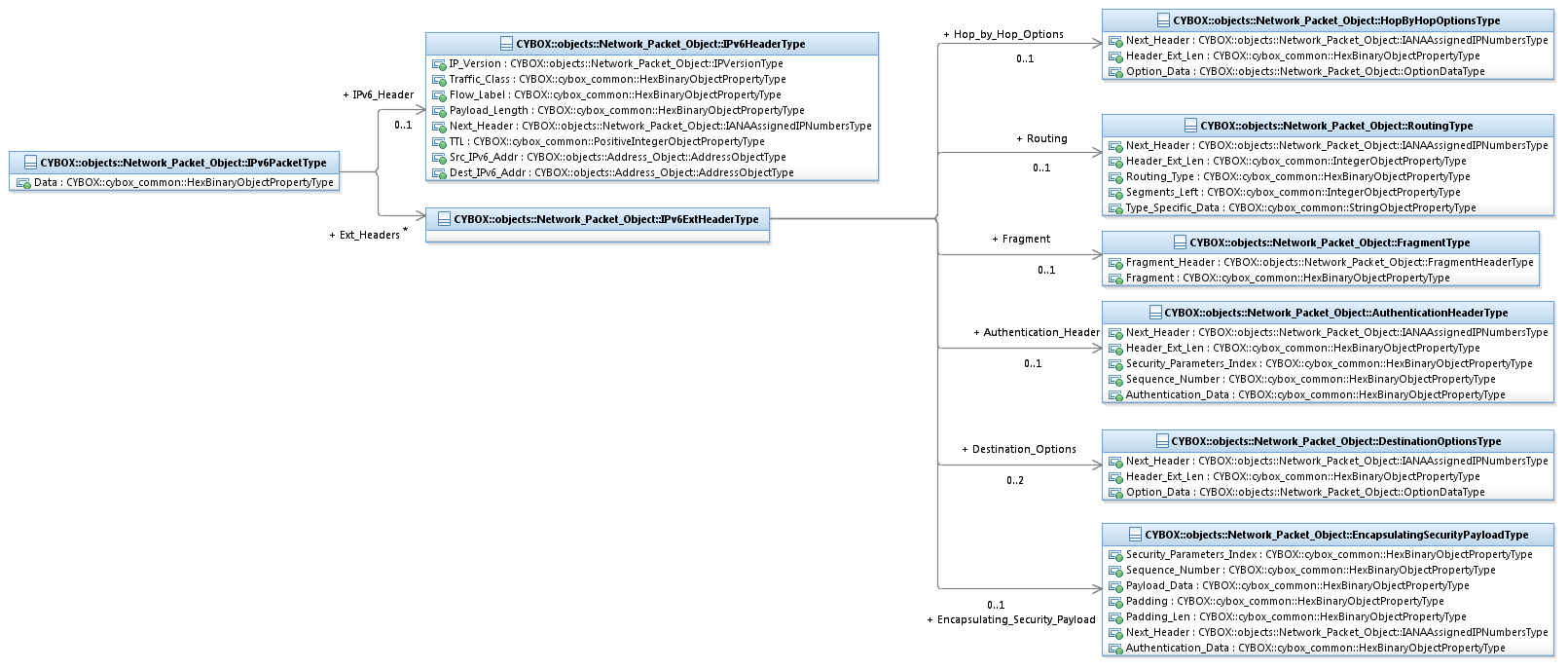


Figure 3‑3. UML diagram for the IPv6PacketType class

The property table of the IPv6PacketType class is given in **Table 3‑30**.

Table 3‑30. Properties of the IPv6PacketType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **IPv6\_Header** | IPv6HeaderType | 0..1 | The IPv6\_Header property specifies the IPv6 header, which is a simplification of the IPv4 header. |
| **Ext\_Headers** | IPv6ExtHeaderType | 0..\* | The Ext\_Headers property specifies the optional internet-layer information which is encoded in separate headers that may be placed between the IPv6 header and the upper-layer header in a packet. |
| **Data** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Data property contains the data portion of an IP packet. Actual property values will probably be specified in the elements of the different network layers, but we provide a field here to capture any data as necessary. |

#### IPv6HeaderType Class

The IPv6HeaderType class specifies the IPv6 header, and is a simplification of the IPv4 header.

The property table of the IPv6HeaderType class is given in **Table 3‑31**.

Table 3‑31. Properties of the IPv6HeaderType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **IP\_Version** | IPVersionType | 0..1 | The IP\_Version property specifies the 4-bit Internet Protocol version number. For this class, the version is always specified using the enumeration literal IPv6(6). |
| **Traffic\_Class** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Traffic\_Class property specifies the 8-bit traffic class. Available for use by originating nodes and/or forwarding routers to identify and distinguish between different classes or priorities of IPv6 packets. See <http://tools.ietf.org/html/rfc2460#section-7> for more information. |
| **Flow\_Label** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Flow\_Label property specifies the 20-bit flow label. Used by a source to label sequences of packets for which it requests special handling by the IPv6 routers, such as non-default quality of service. See <http://tools.ietf.org/html/rfc2460#section-6> for more information. |
| **Payload\_Length** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Payload\_Length property specifies the length of the IPv6 payload (the rest of the packet following the IPv6 header) in octets as a 16-bit unsigned integer. Any extension headers are considered part of the payload. |
| **Next\_Header** | IANAAssignedIPNumbersType | 0..1 | The Next\_Header property specifies the type of header immediately following the IPv6 header. Uses the same values as the IPv4 protocol property. |
| **TTL** | cyboxCommon:  PositiveIntegerObjectPropertyType | 0..1 | The TTL property (TTL/hop limit) specifies how many times a packet can be forwarded, as an 8-bit unsigned integer. |
| **Src\_IPv6\_Addr** | AddressObj:AddressObjectType | 0..1 | The Src\_IPv6\_Addr property specifies the 128-bit address of the originator of the packet. |
| **Dest\_IPv6\_Addr** | AddressObj:AddressObjectType | 0..1 | The Dest\_IPv6\_Addr property specifies the 128-bit address of the intended recipient of the packet. |

#### IPv6ExtHeaderType Class

The IPv6ExtHeaderType class characterizes the optional internet-layer (IPv6) information that is encoded in separate headers. It is placed between the IPv6 header and the upper-layer header in a packet. An IPv6 packet may carry zero, one, or more extension headers. Each extension header is specified in a separate property as described in the table below. See <http://tools.ietf.org/html/rfc2460> for more information.

The property table of the IPv6ExtHeaderType class is given in **Table 3‑32**.

Table 3‑32. Properties of the IPv6ExtHeaderType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Hop\_by\_Hop\_Options** | HopByHopOptionsType | 0..1 | The Hop\_by\_Hop\_Options property specifies the header which is used to carry optional information that must be examined by every node along a packet's delivery path. It carries a variable number of type-length-value (TLV) encoded options. |
| **Routing** | RoutingType | 0..1 | The Routing property specifies the header which is used by an IPv6 source to list one or more intermediate nodes to be "visited" on the way to a packet's destination. |
| **Fragment** | FragmentType | 0..1 | The Fragment property specifies the header which is used by an IPv6 source to send a packet larger than would fit in the path MTU. A fragment packet begins with an unfragmentable part consisting of the IPv6 header plus all extension headers up to and including the routing header. We don't include it for this property because the data is already stored in other properties. We provide the properties necessary for the Fragmentable Part. |
| **Destination\_Options** | DestinationOptionsType | 0..2 | The Destination\_Options property specifies the header which is used to carry optional information that needs to be examined only by a packet's destination node(s). |
| **Authentication\_Header** | AuthenticationHeaderType | 0..1 | The Authentication\_Header property specifies the header which is used for connectionless integrity and data origin authentication for IP datagrams and the protection against replays. See <http://tools.ietf.org/html/rfc2402> for more information. |
| **Encapsulating\_**  **Security\_Payload** | EncapsulatingSecurityPayloadType | 0..1 | The Encapsulating\_Security\_Payload property (ESP) specifies the header which is used for confidentiality, data origin authentication, connectionless integrity, anti-replay service (a form of partial sequence integrity), and limited traffic flow confidentiality. See <http://tools.ietf.org/html/rfc2406> for more information. |

#### HopByHopOptionsType Class

The HopByHopOptionsType class characterizes the IPv6 Hop-by-Hop Options header which is used to carry optional information that must be examined by every node along a packet's delivery path.

The property table of the HopByHopOptionsType class is given in **Table 3‑33**.

Table 3‑33. Properties of the HopByHopOptionsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Next\_Header** | IANAAssignedIPNumbersType | 0..1 | The Next\_Header property specifies the type of header immediately following the Hop-by-Hop Options header. Uses the same values as the IPv4 Protocol property. |
| **Header\_Ext\_Len** | cyboxCommon: HexBinaryObjectPropertyType | 0..1 | The Header\_Ext\_Len property specifies the length of the Hop-by-Hop Options header in 8-octet units, not including the first 8 octets. |
| **Option\_Data** | OptionDataType | 0..\* | The Option\_Data property specifies a variable-length property, of length such that the complete Hop-by-Hop Options header is an integer multiple of 8 octets long. Contains one or more type-length-value (TLV)-encoded options. |

##### OptionDataType Class

The OptionDataType class characterizes the variable-length fields associated with IPv6 extension headers (the Hop-by-Hop Options header and the Destination Options header). Contains one or more class-length-value (TLV)-encoded options.

The property table of the OptionDataType class is given in **Table 3‑34**.

Table 3‑34. Properties of the OptionDataType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Option\_Type** | IPv6OptionType | 0..1 | The Option\_Type property specifies the type of option. This 8-bit Option Type identifier is internally encoded such that different bits have different meanings. |
| **Option\_Data\_Len** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Option\_Data\_Len property specifies the length of the Option Data property of this option, in octets. |
| **Pad1** | Pad1Type | 0..1 | The Pad1 property specifies the option which is used to insert one octet of padding into the Options area of a header. The Pad1 option does not have length and value properties. |
| **PadN** | PadNType | 0..1 | The PadN property specifies the option which is used to insert two or more octets of paddings into the Options area of a header. |

##### Pad1Type Class

The Pad1Type class specifies whether one octet of padding is inserted into the Options area of a header. The Pad1 option type does not have length and value fields.

The property table of the Pad1Type class is given in **Table 3‑35**.

Table 3‑35. Properties of the Pad1Type class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Octet** | cyboxCommon:  HexBinaryObjectPropertyType | 1 | The Octet property specifies whether the Pad1 option is used and also serves as the single octet of padding. |

##### PadNType Class

The PadNType class specifies whether two or more octets of padding are inserted into the Options area of a header.

The property table of the PadNType class is given in **Table 3‑36**.

Table 3‑36. Properties of the PadNType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Octet** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Octet property specifies the PadN option. |
| **Option\_Data\_Length** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Option\_Data\_Length property specifies the length of the padding. For N octets of padding, the Option\_Data\_Length property contains the value N-2. |
| **Option\_Data** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Option\_Data property specifies the actual padding. It consists of N-2 zero-valued octets. |

#### RoutingType Class

The RoutingType class specifies the properties of the Routing header, which is used by an IPv6 source to list one or more intermediate nodes to be "visited" on the way to a packet's destination. See <http://tools.ietf.org/html/rfc2460> for more information.

The property table of the RoutingType class is given in **Table 3‑37**.

Table 3‑37. Properties of the RoutingType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Next\_Header** | IANAAssignedIPNumbersType | 0..1 | The Next\_Header property specifies the type of header immediately following the Routing header. Uses the same values as the IPv4 Protocol property. |
| **Header\_Ext\_Len** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Header\_Ext\_Len property specifies the length of the Routing header in 8-octet units, not including the first 8 octets. |
| **Routing\_Type** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Routing\_Type property specifies the 8-bit identifiers of a particular Routing header variant. Further definition will be added as required. |
| **Segments\_Left** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Segments\_Left property specifies the number of route segments remaining, i.e., number of explicitly listed intermediate nodes still to be visited before reaching the final destination. |
| **Type\_Specific\_Data** | cyboxCommon:  StringObjectPropertyType | 0..1 | The Type\_Specific\_Data property is a variable length property of format determined by the Routing Type. |

#### FragmentType Class

The FragmentType class specifies the properties of the Fragment header, which is used by an IPv6 source to send a packet larger than would fit in the path MTU. See <http://tools.ietf.org/html/rfc2460> for more information.

The property table of the FragmentType class is given in **Table 3‑38**.

Table 3‑38. Properties of the FragmentType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Fragment\_Header** | FragmentHeaderType | 0..1 | The Fragment\_Header property specifies, for each fragment, a header containing next header information, the offset of the fragment, an M flag specifying whether or not it is the last fragment, and an identification value. |
| **Fragment** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Fragment property specifies the fragment of the packet that corresponds to the fragment header. The length of the fragment must fit with the MTU of the path to the packets' destination. |

##### FragmentHeaderType Class

The FragmentHeaderType class characterizes each fragment with a header containing next header information, the offset of the fragment, an M flag specifying whether or not it is the last fragment, and an identification value.

The property table of the FragmentHeaderType class is given in **Table 3‑39**.

Table 3‑39. Properties of the FragmentHeaderType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Next\_Header** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Next\_Header property specifies the type of header immediately following the Fragment header. Uses the same values as the IPv4 Protocol property. |
| **Fragment\_Offset** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Fragment\_Offset property specifies a 13-bit unsigned integer which is the offset, in 8-octet units, of the data following this header relative to the start of the Fragmentable Part or the original packet. |
| **M\_Flag** | MFlagType | 0..1 | The M\_Flag property specifies whether this is the last fragment or whether there are more fragments. |
| **Identification** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Identification property specifies a 32-bit Identification value for the packet generated by the source node. |

##### MFlagType Data Type

The MFlagType data type specifies whether there are more fragments. Its core value SHOULD be a literal found in the MFlagTypeEnum enumeration. Its base type is the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

#### DestinationOptionsType Class

The DestinationOptionsType class specifies the properties for the IPv6 Destination Options header which is used to carry optional information that needs to be examined only by a packet's destination node(s).

The property table of the DestinationOptionsType class is given in **Table 3‑40**.

Table 3‑40. Properties of the DestinationOptionsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Next\_Header** | IANAAssignedIPNumbersType | 0..1 | The Next\_Header property specifies the type of header immediately following the Destination\_Options options header. Uses the same values as the IPv4 Protocol property. |
| **Header\_Ext\_Len** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Header\_Ext\_Len property specifies the length of the Destination Options header in 8-octet units, not including the first 8 octets. |
| **Option\_Data** | OptionDataType | 0..\* | The Option\_Data property specifies a variable-length property of length such that the complete Destinations Options header is an integer multiple of 8 octets long. Contains one or more type-length-value (TLV)-encoded options. |

#### AuthenticationHeaderType Class

The AuthenticationHeaderType class is used to provide connectionless integrity and data origin authentication for IP datagrams and to provide protection against replays. See <http://tools.ietf.org/html/rfc2402> for more information.

The property table of the AuthenticationHeaderType class is given in **Table 3‑41**.

Table 3‑41. Properties of the AuthenticationHeaderType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Next\_Header** | IANAAssignedIPNumbersType | 0..1 | The Next\_Header property specifies the type of header immediately following the Authentication header. Uses the same values as the IPv4 Protocol property. |
| **Header\_Ext\_Len** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Header\_Ext\_Len property is an 8-bit property specifying the length of the AH in 32-bit words. |
| **Security\_Parameters\_Index** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Security\_Parameters\_Index (SPI) property is an arbitrary 32-bit value that, in combination with the destination IP address and security protocol (AH), uniquely identifies the Security Association for this datagram. The set of SPI values in the range 1 through 255 are reserved by the Internet Assigned Numbers Authority (IANA) for future use. |
| **Sequence\_Number** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Sequence\_Number property specifies the unsigned 32-bit property containing a monotonically increasing counter value (sequence number). |
| **Authentication\_Data** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Authentication\_Data property specifies a variable-length property that contains the Integrity Check Value (ICV) for this packet. The field must be an integer multiple of 32 bits in length. |

#### EncapsulatingSecurityPayloadType Class

The EncapsulatingSecurityPayloadType (ESP) class is used to provide confidentiality, data origin authentication, connectionless integrity, an anti-replay service (a form of partial sequence integrity), and limited traffic flow confidentiality. See [http:/tools.ietf.org/html/rfc2406](http://tools.ietf.org/html/rfc2406) for more information.

The property table of the EncapsulatingSecurityPayloadType class is given in **Table 3‑42**.

Table 3‑42. Properties of the EncapsulatingSecurityPayloadType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Security\_Parameters\_Index** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Security\_Parameters\_Index (SPI) property is an arbitrary 32-bit value that, in combination with the destination IP address and security protocol (ESP), uniquely identifies the Security Association for this datagram. |
| **Sequence\_Number** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Sequence\_Number property is an unsigned 32-bit integer that contains a monotonically increasing counter value (sequence number). |
| **Payload\_Data** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Payload\_Data property specifies a variable-length property containing data described by the Next\_Header property. |
| **Padding** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Padding property specifies the contents that can be used for various reasons, such as to fill in the plaintext as required by an encryption algorithm or to conceal the actual length of the payload. |
| **Padding\_Len** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Padding\_Len property specifies the number of pad bytes immediately preceding it. Range is 0-255, where a value of zero indicates that no padding bytes are present. |
| **Next\_Header** | IANAAssignedIPNumbersType | 0..1 | The Next\_Header property specifies the type data contained in the Payload\_Data property. Uses the same values as the IPv4 Protocol field. |
| **Authentication\_Data** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Authentication\_Data property is a variable-length property containing an Integrity Check Value (ICV) computed over the ESP packet minus the Authentication Data. |

#### IPv6OptionType Class

The IPv6OptionType class characterizes the meaning of each bit of the 8-bit IPv6 Options.

The property table of the IPv6OptionType class is given in **Table 3‑43**.

Table 3‑43. Properties of the IPv6OptionType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Do\_Not\_Recogn\_Action** | IPv6DoNotRecogActionType | 0..1 | The Do\_Not\_Recogn\_Action property specifies the action to be taken if the processing IPv6 nodes do not recognize the Option Type. This information is internally encoded in the Option Type identifier (highest-order two bits) such that their highest-order two bits specify the action that must be taken. |
| **Packet\_Change** | IPv6PacketChangeType | 0..1 | The Packet\_Change property specifies the third highest order bit of the Option Data and indicates whether or not the Option Data of that option can change en-route to the packet's final destination. |
| **Option\_Byte** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Option\_Byte property may be used to specify the actual Option Type byte, with no explicit meaning attached. Meaning/interpretation provided by the Do\_Not\_Recogn\_Action and Packet\_Change properties. |

#### IPv6DoNotRecogActionType Data Type

The IPv6DoNotRecogActionType data type specifies possible actions when an option is not recognized. Its core value SHOULD be a literal found in the IPv6DoNotRecogActionTypeEnum enumeration. Its base type is the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

#### IPv6PacketChangeType Data Type

The IPV6PacketChangeType data type specifies whether a packet has changed. Its core value SHOULD be a literal found in the IPv6PacketChangeTypeEnum enumeration. Its base type is the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

### ICMPv4PacketType Class

The ICMPv4PacketType class characterizes the ICMP (v4), which is used to send error messages (e.g., a datagram cannot reach its destination), informational messages (e.g., timestamp information), or a traceroute message. See <http://www.networksorcery.com/enp/protocol/icmp.htm> for more information.

In CybOX 2.1.1, all of the properties of the ICMPv4PacketType class, except ICMPv4\_Header, are mutually exclusive, i.e., only one property can be populated. This restriction is based on the fact that the classes ICMPv4ErrorMessageType, ICMPv4InfoMessageType and ICMPv4TracerouteType can be modelled as subclasses of the ICMPv4PacketType class. In future releases, it will probably be modelled that way.

The UML diagram corresponding to the ICMPv4PacketType class is shown in **Figure 3‑4**.

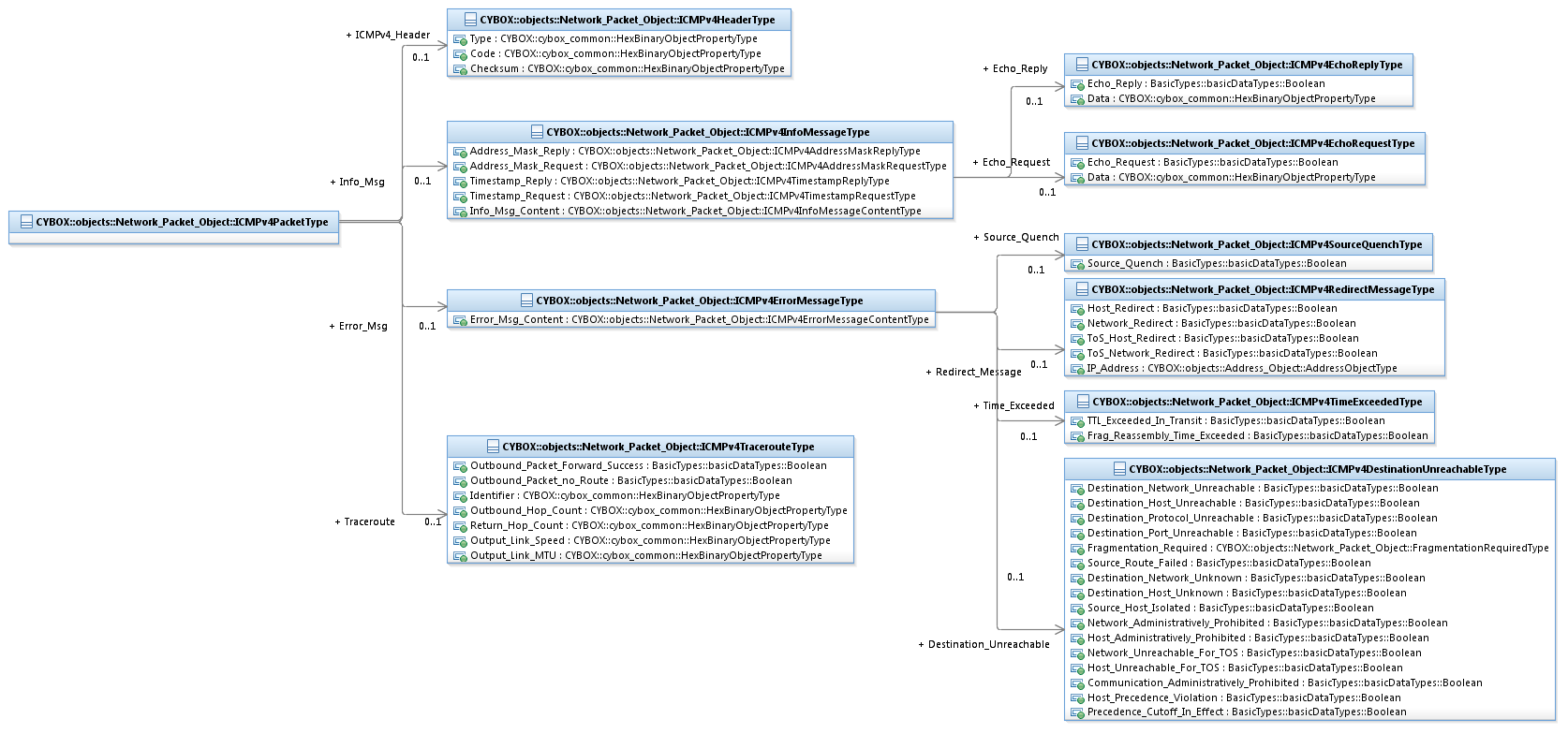


Figure 3‑4. UML diagram for the ICMPv4PacketType class

The property table of the ICMPv4PacketType class is given in **Table 3‑44**.

Table 3‑44. Properties of the ICMPv4PacketType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **ICMPv4\_Header** | ICMPv4HeaderType | 0..1 | The ICMPv4\_Header property specifies the type and code properties. |
| **Error\_Msg** | ICMPv4ErrorMessageType | 0..1 | The Error\_Msg property is used to specify an ICMPv4ErrorMessageType form of the ICMPv4PacketType class.  Only one of the Error\_Msg, Info\_Msg and Traceroute properties can be populated. |
| **Info\_Msg** | ICMPv4InfoMessageType | 0..1 | The Info\_Msg property is used to specify an ICMPv4InfoMessageType form of the ICMPv4PacketType class.    Only one of the Error\_Msg, Info\_Msg and Traceroute properties can be populated. |
| **Traceroute** | ICMPv4TracerouteType | 0..1 | The Traceroute property is used to specify an ICMPv4TracerouteType form of the ICMPv4PacketType class.    Only one of the Error\_Msg, Info\_Msg and Traceroute properties can be populated. |

#### ICMPv4HeaderType Class

The ICMPv4HeaderType class specify the actual ICMP header bytes corresponding to the ICMP class, ICMP code, and to the checksum.

The property table of the ICMPv4HeaderType class is given in **Table 3‑45**.

Table 3‑45. Properties of the ICMPv4HeaderType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Type** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Type property specifies the format of the ICMP message. |
| **Code** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Code property specifies the code of the ICMP message. |
| **Checksum** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Checksum property specifies the checksum (16 bits) of the ICMP message. |

#### ICMPv4ErrorMessageType Class

The ICMPv4ErrorMessageType class specifies ICMP error messages, which include destination unreachable messages, source quench messages, redirect messages, and time exceeded messages.

In CybOX 2.1.1, all of the properties of the ICMPv4ErrorMessageType class, except Error\_Msg\_Content, are mutually exclusive, i.e., only one property can be populated. This restriction is based on the fact that the classes ICMPv4DestinationUnreachableType, ICMPv4SourceQuenchType, ICMPv4RedirectMessageType and ICMPv4TimeExceededType can be modelled as subclasses of the ICMPv4ErrorMessageType class. In future releases, it will probably be modelled that way.

The property table of the ICMPv4ErrorMessageType class is given in **Table 3‑46**.

Table 3‑46. Properties of the ICMPv4ErrorMessageType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Error\_Msg\_Content** | ICMPv4ErrorMessageContentType | 0..1 | The Error\_Msg\_Content property specifies content common to all ICMP error message types are defined here. Properties that are specific to individual message types are defined separately under each message type. |
| **Destination\_Unreachable** | ICMPv4DestinationUnreachableType | 0..1 | The Destination\_Unreachable property specifies a destination unreachable message, which is generated by the host or its inbound gateway to inform the client that the destination is unreachable for some reason. See <http://en.wikipedia.org/wiki/ICMP_Destination_Unreachable> for more information.  Only one of the Destination\_Unreachable, Source\_Quench, Redirect\_Message and Time\_Exceeded properties can be populated. |
| **Source\_Quench** | ICMPv4SourceQuenchType | 0..1 | The Source\_Quench property specifies a source quench message that requests the sender decrease the rate of messages sent to a router or host. This message may be generated if a router or host does not have sufficient buffer space to process the request or may occur if the router or host buffer is approaching its limit. See <http://en.wikipedia.org/wiki/ICMP_Source_Quench> for more information.  Only one of the Destination\_Unreachable, Source\_Quench, Redirect\_Message and Time\_Exceeded properties can be populated. |
| **Redirect\_Message** | ICMPv4RedirectMessageType | 0..1 | The Redirect\_Message property specifies a redirect message which is used to send data packets on an alternative route. This ICMP redirect message informs a host to update its routing information.  Only one of the Destination\_Unreachable, Source\_Quench, Redirect\_Message and Time\_Exceeded properties can be populated. |
| **Time\_Exceeded** | ICMPv4TimeExceededType | 0..1 | The Time\_Exceeded property specifies a time exceeded message, which is generated by a gateway to inform the source of a datagram that the datagram has been discarded due to the time to live property reaching zero. A time exceeded message may also be sent by a host if it fails to reassemble a fragmented datagram within its time limit. See <http://en.wikipedia.org/wiki/ICMP_Time_Exceeded> for more information.  Only one of the Destination\_Unreachable, Source\_Quench, Redirect\_Message and Time\_Exceeded properties can be populated. |

#### ICMPv4ErrorMessageContentType Class

The ICMPv4ErrorMessageContentType class specifies properties common to all types of ICMPv4 error messages.

The property table of the ICMPv4ErrorMessageContentType class is given in **Table 3‑47**.

Table 3‑47. Properties of the ICMPv4ErrorMessageContentType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **IP\_Header** | IPv4HeaderType | 0..1 | The IP\_Header property specifies the IP header from the original datagram. |
| **First\_Eight\_Bytes** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The First\_Eight\_Bytes property specifies the first 8 bytes of the original datagram's data. |

##### ICMPv4DestinationUnreachableType Class

The ICMPv4DestinationUnreachableType class specifies the Destination Unreachable error message; ICMP class=3.

In CybOX 2.1.1, all of the properties of the ICMPv4DestinationUnreachableType class are mutually exclusive, i.e., only one property can be populated. This restriction is based on the fact each property is a Boolean value to indicate if the message is the one of sixteen types of Destination Unreachable messages. In future releases, this could be modelled as using one property for the subtype (code) and an enumeration.

The property table of the ICMPv4DestinationUnreachableType class is given in **Table 3‑48**.

Table 3‑48. Properties of the ICMPv4DestinationUnreachableType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Destination\_Network\_Unreachable** | basicDataTypes:Boolean | 0..1 | The Destination\_Network\_Unreachable property specifies whether the subtype of the message is “destination network unreachable” (code=0). |
| **Destination\_Host\_Unreachable** | basicDataTypes:Boolean | 0..1 | The Destination\_Host\_Unreachable property specifies whether the subtype of the message is “destination host unreachable” (code=1). |
| **Destination\_Protocol\_Unreachable** | basicDataTypes:Boolean | 0..1 | The Destination\_Protocol\_Unreachable property specifies whether the subtype of the message is “destination protocol unreachable” (code=2). |
| **Destination\_Port\_Unreachable** | basicDataTypes:Boolean | 0..1 | The Destination\_Port\_Unreachable property specifies whether the subtype of the message is “destination port unreachable” (code=3). |
| **Fragmentation\_Required** | FragmentationRequiredType | 0..1 | The Fragmentation\_Required property specifies whether the subtype of the message is “fragmentation required” (code=4). This property has an additional field (Next-Hop MTU), as well as a boolean value indicating this subtype. |
| **Source\_Route\_Failed** | basicDataTypes:Boolean | 0..1 | The Source\_Route\_Failed property specifies whether the subtype of the message is “source route failed” (code=5). |
| **Destination\_Network\_Unknown** | basicDataTypes:Boolean | 0..1 | The Destination\_Network\_Unknown property specifies whether the subtype of the message is “destination network unknown” (code=6). |
| **Destination\_Host\_Unknown** | basicDataTypes:Boolean | 0..1 | The Destination\_Host\_Unknown property specifies whether the subtype of the message is “destination host unknown” (code=7). |
| **Source\_Host\_Isolated** | basicDataTypes:Boolean | 0..1 | The Source\_Host\_Isolated property specifies whether the subtype of the message is “source host isolated” (code=8). |
| **Network\_Administratively\_**  **Prohibited** | basicDataTypes:Boolean | 0..1 | The Network\_Administratively\_Prohibited property specifies whether the subtype of the message is “network administratively prohibited” (code=9). |
| **Host\_Administratively\_Prohibited** | basicDataTypes:Boolean | 0..1 | The Host\_Administratively\_Prohibited property specifies whether the subtype of the message is “host administratively prohibited” (code=10). |
| **Network\_Unreachable\_For\_TOS** | basicDataTypes:Boolean | 0..1 | The Network\_Unreachable\_For\_TOS property specifies whether the subtype of the message is “network unreachable for TOS” (code=11). |
| **Host\_Unreachable\_For\_TOS** | basicDataTypes:Boolean | 0..1 | The Host\_Unreachable\_For\_TOS specifies whether the subtype of the message is “host unreachable for TOS” (code=12). |
| **Communication\_Administratively\_**  **Prohibited** | basicDataTypes:Boolean | 0..1 | The Communication\_Administratively\_Prohibited property specifies whether the subtype of the message is “communication administratively prohibited” (code=13). |
| **Host\_Precedence\_Violation** | basicDataTypes:Boolean | 0..1 | The Host\_Precedence\_Violation property specifies whether the subtype of the message is “host precedence violation” (code=14). |
| **Precedence\_Cutoff\_In\_Effect** | basicDataTypes:Boolean | 0..1 | The Precedence\_Cutoff\_In\_Effect property specifies whether the subtype of the message is “precedence cutoff in effect” (code=15). |

##### FragmentationRequiredType Class

The FragmentationRequiredType class further specifies an ICMP destination unreachable (class=3) message as the subtype fragmentation required message (code=4) by providing a Next-Hop MTU field.

The property table of the FragmentationRequiredType class is given in **Table 3‑49**.

Table 3‑49. Properties of the FragmentationRequiredType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Fragmentation\_Required** | basicDataTypes:Boolean | 0..1 | The Fragmentation\_Required property specifies whether the subtype of the destination unreachable ICMP message is "fragmentation required". |
| **Next\_Hop\_MTU** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Next\_Hop\_MTU property contains the MTU of the next-hop network when a code 4 error (fragmentation required) occurs. |

##### ICMPv4SourceQuenchType Class

The ICMPv4SourceQuenchType class specifies the Source Quench (congestion control) error message; ICMP class=4.

The property table of the ICMPv4SourceQuenchType class is given in **Table 3‑50**.

Table 3‑50. Properties of the ICMPv4SourceQuenchType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Source\_Quench** | basicDataTypes:Boolean | 0..1 | The Source\_Quench property specifies whether the subtype of the error message is a source quench (code=0). |

##### ICMPv4RedirectMessageType Class

The ICMPv4RedirectMessageType class specifies the Redirect Message error message; ICMP class=5.

In CybOX 2.1.1, all of the properties of the ICMPv4RedirectMessageType class except IP\_Address are mutually exclusive, i.e., only one property can be populated. This restriction is based on the fact each property is a Boolean value to indicate if the message is the one of four types of Redirect messages. In future releases, this could be modelled as using one property for the subtype (code) and an enumeration.

The property table of the ICMPv4RedirectMessageType class is given in **Table 3‑51**.

Table 3‑51. Properties of the ICMPv4RedirectMessageType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **IP\_Address** | AddressObj:  AddressObjectType | 0..1 | The IP\_Address property specifies the IP address is the 32-bit address of the gateway to which the redirection should be sent. |
| **Network\_Redirect** | basicDataTypes:  Boolean | 0..1 | The Network\_Redirect property specifies whether the subtype of the message is a redirect datagram for the network (code=0).  Only one of the Network\_Redirect, Host\_Redirect, ToS\_Network\_Redirect and ToS\_Host\_Redirect properties can be populated. |
| **Host\_Redirect** | basicDataTypes:  Boolean | 0..1 | The Host\_Redirect property specifies whether the subtype of the message is a redirect datagram for the host (code=1).  Only one of the Network\_Redirect, Host\_Redirect, ToS\_Network\_Redirect and ToS\_Host\_Redirect properties can be populated. |
| **ToS\_Network\_Redirect** | basicDataTypes:  Boolean | 0..1 | The ToS\_Network\_Redirect property specifies whether the subtype of the message is a redirect datagram for the TOS and network (code=2).  Only one of the Network\_Redirect, Host\_Redirect, ToS\_Network\_Redirect and ToS\_Host\_Redirect properties can be populated. |
| **ToS\_Host\_Redirect** | basicDataTypes:  Boolean | 0..1 | The ToS\_Host\_Redirect property specifies whether the subtype of the message is a redirect datagram for the TOS and host (code=3).  Only one of the Network\_Redirect, Host\_Redirect, ToS\_Network\_Redirect and ToS\_Host\_Redirect properties can be populated. |

##### ICMPv4TimeExceededType Class

The ICMPv4TimeExceededType class specifies the Time Exceeded error message; ICMP class=11.

In CybOX 2.1.1, the properties, TTL\_Exceeded\_In\_Transit and Frag\_Reassembly\_Time\_Exceeded are mutually exclusive, i.e., only one property can be populated. This restriction is based on that there are two different types of time exceeded messages. In future releases, it will probably be replaced with one property and a corresponding enumeration.

The property table of the ICMPv4TimeExceededType class is given in **Table 3‑52**.

Table 3‑52. Properties of the ICMPv4TimeExceededType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **TTL\_Exceeded\_In\_Transit** | basicDataTypes:  Boolean | 0..1 | The TTL\_Exceeded\_In\_Transit property specifies that the time-to-live was exceeded in transit (code=0). |
| **Frag\_Reassembly\_**  **Time\_Exceeded** | basicDataTypes:  Boolean | 0..1 | The Frag\_Reassembly\_Time\_Exceeded property specifies that the fragment reassembly time was exceeded (code=1). |

#### ICMPv4InfoMessageType Class

The ICMPv4InfoMessageType class specifies ICMP informational messages, which include echo request/reply, timestamp request/reply, and address mask request/reply.

In CybOX 2.1.1, all of the properties of the ICMPv4InfoMessageType class, except Info\_Msg\_Content, are mutually exclusive, i.e., only one property can be populated. This restriction is based on the fact that the classes ICMPv4EchoReplyType, ICMPv4EchoRequestType, ICMPv4TimestampRequestType, ICMPv4TimestampReplyType, ICMPv4AddressMaskRequestType and ICMPv4AddressMaskReplyType can be modelled as subclasses of the ICMPv4InfoMessageType class. In future releases, it will probably be modelled that way.

The property table of the ICMPv4InfoMessageType class is given in **Table 3‑53**.

Table 3‑53. Properties of the ICMPv4InfoMessageType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Info\_Msg\_Content** | ICMPv4InfoMessageContentType | 0..1 | The Info\_Msg\_Content property specifies properties that are common to all ICMP informational. Properties that are specific to individual messages are defined separately under each message type. |
| **Echo\_Reply** | ICMPv4EchoReplyType | 0..1 | The Echo\_Reply property specifies an echo reply message (type=0). These messages are also known as "ping".  Only one of the Echo\_Reply, Echo\_Request, Timestamp\_Request, Timestamp\_Reply, Address\_Mask\_Request and Address\_Mask\_Reply properties can be populated. |
| **Echo\_Request** | ICMPv4EchoRequestType | 0..1 | The Echo\_Request property specifies an echo reply message (type=8). These messages are also known as "ping".  Only one of the Echo\_Reply, Echo\_Request, Timestamp\_Request, Timestamp\_Reply, Address\_Mask\_Request and Address\_Mask\_Reply properties can be populated. |
| **Timestamp\_Request** | ICMPv4TimestampRequestType | 0..1 | The Timestamp\_Request property specifies a timestamp request used for time synchronization (type=13).  Only one of the Echo\_Reply, Echo\_Request, Timestamp\_Request, Timestamp\_Reply, Address\_Mask\_Request and Address\_Mask\_Reply properties can be populated. |
| **Timestamp\_Reply** | ICMPv4TimestampReplyType | 0..1 | The Timestamp\_Reply property specifies a timestamp reply which replies to a timestamp request message (type=14).  Only one of the Echo\_Reply, Echo\_Request, Timestamp\_Request, Timestamp\_Reply, Address\_Mask\_Request and Address\_Mask\_Reply properties can be populated. |
| **Address\_Mask\_Request** | ICMPv4AddressMaskRequestType | 0..1 | The Address\_Mask\_Request property specifies an address mask request which is a query message normally sent by a host to a router in order to obtain an appropriate subnet mask (type=17).  Only one of the Echo\_Reply, Echo\_Request, Timestamp\_Request, Timestamp\_Reply, Address\_Mask\_Request and Address\_Mask\_Reply properties can be populated. |
| **Address\_Mask\_Reply** | ICMPv4AddressMaskReplyType | 0..1 | The Address\_Mask\_Reply property specifies an address mask reply which is used to reply to an address mask request message with an appropriate subnet mask (type=18).  Only one of the Echo\_Reply, Echo\_Request, Timestamp\_Request, Timestamp\_Reply, Address\_Mask\_Request and Address\_Mask\_Reply properties can be populated. |

##### ICMPv4InfoMessageContentType Class

The ICMPv4InfoMessageContentType class specifies properties common to all types of ICMPv4 informational messages.

The property table of the ICMPv4InfoMessageContentType class is given in **Table 3‑54**.

Table 3‑54. Properties of the ICMPv4InfoMessageContentType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Identifier** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Identifier property specifies a 16-bit identifier, which is combined with the sequence number, and called the "quench" for echo reply and echo request. |
| **Sequence\_Number** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Sequence\_Number property specifies a 16-bit sequence number. The identifier and sequence number can be used by the client to match the reply with the request that caused the reply. |

##### ICMPv4EchoReplyType Class

the ICMPv4EchoReplyType class specifies the echo reply v4 informational message (used to ping); ICMP class=0.

The property table of the ICMPv4EchoReplyType class is given in **Table 3‑55**.

Table 3‑55. Properties of the ICMPv4EchoReplyType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Echo\_Reply** | basicDataTypes:Boolean | 0..1 | The Echo\_Reply property specifies whether the subtype of the message is an echo reply message (code=0). |
| **Data** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Data property specifies the data of the different kind of answers given with an ICMP Echo Reply message. It can be of arbitrary length (but less than the MTU of the network). |

##### ICMPv4EchoRequestType Class

The ICMPv4EchoRequestType class specifies the echo request informational message (used to ping); ICMP class=8.

The property table of the ICMPv4EchoRequestType class is given in **Table 3‑56**.

Table 3‑56. Properties of the ICMPv4EchoRequestType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Echo\_Request** | basicDataTypes:Boolean | 0..1 | The Echo\_Request property specifies whether the subtype of the message is an echo request message (code=8). |
| **Data** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Data property specifies the data of the different kind of answers given with an ICMP Echo Request message. It can be of arbitrary length (but less than the MTU of the network). |

##### ICMPv4TimestampRequestType Class

The ICMPv4TimestampRequestType class specifies the timestamp request informational message; ICMP class=13.

The property table of the ICMPv4TimestampRequestType class is given in **Table 3‑57**.

Table 3‑57. Properties of the ICMPv4TimestampRequestType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Timestamp** | basicDataTypes:Boolean | 0..1 | The Timestamp property specifies whether the subtype of the message is a timestamp request message (code=13). |
| **Originate\_Timestamp** | cyboxCommon:  UnsignedIntegerObjectPropertyType | 0..1 | The Originate\_Timestamp property specifies number of milliseconds since midnight UT, in 32-bits. The originate timestamp is the time the sender last touched the message before sending it. If the time is not available in milliseconds or cannot be provided with respect to midnight UT, then any time can be inserted in a timestamp provided the high order bit of the timestamp is also set to indicate this non-standard value. |

##### ICMPv4TimestampReplyType Class

The ICMPv4TimestampReplyType class specifies the timestamp reply informational message; ICMP class=14.

The property table of the ICMPv4TimestampReplyType class is given in **Table 3‑58**.

Table 3‑58. Properties of the ICMPv4TimestampReplyType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Timestamp\_Reply** | basicDataTypes:Boolean | 0..1 | The Timestamp\_Reply property specifies whether the subtype of the message is a timestamp reply message (code=14). |
| **Originate\_Timestamp** | cyboxCommon:  UnsignedIntegerObjectPropertyType | 0..1 | The Originate\_Timestamp property specifies the timestamp of the time that the sender last touched the message before sending it. If the time is not available in milliseconds or cannot be provided with respect to midnight UT, then any time can be inserted in a timestamp provided the high order bit of the timestamp is also set to indicate this non-standard value. |
| **Receive\_Timestamp** | cyboxCommon:  UnsignedIntegerObjectPropertyType | 0..1 | The Receive\_Timestamp property specifies the timestamp of the time the echoer first touched the message on receipt. If the time is not available in milliseconds or cannot be provided with respect to midnight UT, then any time can be inserted in a timestamp provided the high order bit of the timestamp is also set to indicate this non-standard value. |
| **Transmit\_Timestamp** | cyboxCommon:  UnsignedIntegerObjectPropertyType | 0..1 | The Transmit\_Timestamp property specifies the timestamp of the time the echoer last touched the message on sending it. If the time is not available in milliseconds or cannot be provided with respect to midnight UT, then any time can be inserted in a timestamp provided the high order bit of the timestamp is also set to indicate this non-standard value. |

##### ICMPv4AddressMaskRequestType Class

The ICMPv4AddressMaskRequestType class specifies the address mask request informational message; ICMP class=17.

The property table of the ICMPv4AddressMaskRequestType class is given in **Table 3‑59**.

Table 3‑59. Properties of the ICMPv4AddressMaskRequestType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Address\_Mask\_Request** | basicDataTypes:Boolean | 0..1 | The Address\_Mask\_Request property specifies whether the subtype of the message is an address mask request message (code=17). |
| **Address\_Mask** | AddressObj:  AddressObjectType | 0..1 | The Address\_Mask property specifies the address mask. It can be set to zero (as opposed to an address mask reply message, in which case it should be set to the subnet mask). |

##### ICMPv4AddressMaskReplyType Class

The ICMPv4AddressMaskReplyType class specifies the address mask informational message; ICMP class=18.

The property table of the ICMPv4AddressMaskReplyType class is given in **Table 3‑60**.

Table 3‑60. Properties of the ICMPv4AddressMaskReplyType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Address\_Mask\_Reply** | basicDataTypes:Boolean | 0..1 | The Address\_Mask\_Reply property specifies whether the subtype of the message is an address mask reply message (code=18). |
| **Address\_Mask** | AddressObj:  AddressObjectType | 0..1 | This Address\_Mask property should be set to the subnet mask. |

#### ICMPv4TracerouteType Class

The ICMPv4TracerouteType class specifies properties associated with ICMPv4 traceroute message (class =30). See <http://www.networksorcery.com/enp/protocol/icmp/msg30.htm> for more information.

In CybOX 2.1.1, the properties, Outbound\_Packet\_Forward\_Success and Outbound\_Packet\_no\_Route are mutually exclusive, i.e., only one property can be populated. This restriction is based on that there are two different types of traceroute messages. In future releases, it will probably be replaced with one property and a corresponding enumeration.

The property table of the ICMPv4TracerouteType class is given in **Table 3‑61**.

Table 3‑61. Properties of the ICMPv4TracerouteType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Outbound\_Packet\_**  **Forward\_Success** | basicDataTypes:Boolean | 0..1 | The Outbound\_Packet\_Forward\_Success property specifies whether the subtype of the message indicates that the outbound packet was successfully forwarded (code=0).  Only one of the Outbound\_Packet\_Forward\_Success, and Outbound\_Packet\_no\_Route properties can be populated. |
| **Outbound\_Packet\_**  **no\_Route** | basicDataTypes:Boolean | 0..1 | The Outbound\_Packet\_no\_Route property specifies whether the subtype of the message indicates that there was no route for the outbound packet and the packet was discarded (code=1).  Only one of the Outbound\_Packet\_Forward\_Success, and Outbound\_Packet\_no\_Route properties can be populated. |
| **Identifier** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Identifier property specifies the ID number as copied from the ICMP traceroute option of the packet which caused this traceroute message to be sent (not related to the ID number in the IP header). |
| **Outbound\_Hop\_Count** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Outbound\_Hop\_Count property specifies the outbound hop count as copied from the IP traceroute option of the packet which caused this traceroute message to be sent. |
| **Return\_Hop\_Count** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Return\_Hop\_Count property specifies the return hop count as copied from the IP traceroute options of the packet which caused this traceroute message to be sent. |
| **Output\_Link\_Speed** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Output\_Link\_Speed property specifies the speed in bytes per second of the link over which the Outbound/Return Packet will be sent. If this value cannot be determined, the property should be set to zero. |
| **Output\_Link\_MTU** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Output\_Link\_MTU property specifies the MTU in bytes of the link over which the Outbound/Return Packet will be sent. MTU refers to the data portion (includes IP header; excludes datalink header/trailer) of the packet. If this value cannot be determined, this property should be set to zero. |

### ICMPv6PacketType Class

The ICMPv6PacketType class characterizes the ICMP (v4), which is used send error messages (e.g., a datagram cannot reach its destination), informational messages ( e.g., ping). Only the message class defined in <http://tools.ietf.org/html/rfc4443> (ICMP v6) are included; additional message types will be defined as needed. See <http://www.networksorcery.com/enp/protocol/icmpv6.htm> and <http://en.wikipedia.org/wiki/ICMPv6> for more information.

In CybOX 2.1.1, all of the properties of the ICMPv6PacketType class, except ICMPv6\_Header, are mutually exclusive, i.e., only one property can be populated. This restriction is based on the fact that the classes ICMPv6ErrorMessageType and ICMPv4InfoMessageType can be modelled as subclasses of the ICMPv6PacketType class. In future releases, it will probably be modelled that way.

The UML diagram corresponding to the ICMPv6PacketType class is shown in **Figure 3‑5**.

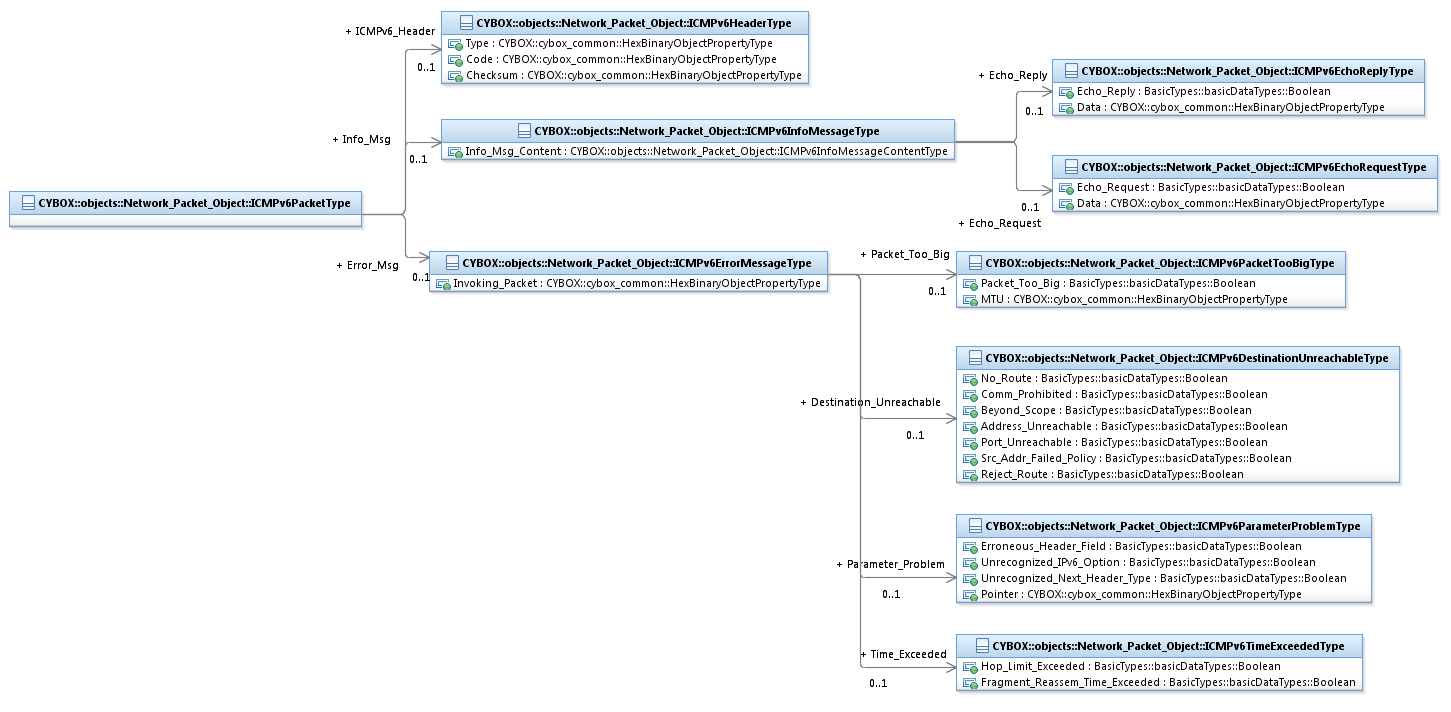


Figure 3‑5. UML diagram for the ICMPv6PacketType class

The property table of the ICMPv6PacketType class is given in **Table 3‑62**.

Table 3‑62. Properties of the ICMPv6PacketType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **ICMPv6\_Header** | ICMPv6HeaderType | 0..1 | The ICMPv6\_Header property specifies the ICMP type, ICMP code, and the checksum. |
| **Error\_Msg** | ICMPv6ErrorMessageType | 0..1 | The Error\_Msg property is used to specify an ICMPv6ErrorMessageType form of the ICMPv6PacketType class.  Only one of the Error\_Msg and Info\_Msg properties can be populated. |
| **Info\_Msg** | ICMPv6InfoMessageType | 0..1 | The Info\_Msg property is used to specify an ICMPv6InfoMessageType form of the ICMPv6PacketType class.  Only one of the Error\_Msg and Info\_Msg properties can be populated. |

#### ICMPv6HeaderType Class

The ICMPv6HeaderType class specifies the header properties: the ICMP class, ICMP code, and the checksum.

The property table of the ICMPv6HeaderType class is given in **Table 3‑63**.

Table 3‑63. Properties of the ICMPv6HeaderType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Type** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Type property specifies the type of the message. Values range from 0 to 127 (high order bit is 0) indicate an error messages; values from 128 to 255 (high order bit is 1) indicate an informational message. |
| **Code** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Code property depends on the message type and provides an additional level of message granularity. |
| **Checksum** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Checksum property specifies the checksum information of an ICMPv6 header. |

#### ICMPv6ErrorMessageType Class

The ICMPv6ErrorMessageType class specifies ICMP v6 error messages, including destination unreachable messages, packet too big messages, time exceeded messages, and parameter problem messages. Type values in the header of ICMP v6 error messages range from 1 to 127.

See <http://tools.ietf.org/html/rfc4443> and <http://tools.ietf.org/html/rfc2463> for more information.

In CybOX 2.1.1, all of the properties of the ICMPv6ErrorMessageType class, except Invoking\_Packet, are mutually exclusive, i.e., only one property can be populated. This restriction is based on the fact that the classes ICMPv6DestinationUnreachableType, ICMPv6PacketTooBigType, ICMPv6TimeExceededType and ICMPv6ParameterProblemType can be modelled as subclasses of the ICMPv6ErrorMessageType class. In future releases, it will probably be modelled that way.

The property table of the ICMPv6ErrorMessageType class is given in **Table 3‑64**.

Table 3‑64. Properties of the ICMPv6ErrorMessageType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Invoking\_Packet** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Invoking\_Packet property specifies as much of invoking packet as possible without the ICMPv6 packet exceeding the minimum IPc6 MTU. |
| **Destination\_Unreachable** | ICMPv6DestinationUnreachableType | 0..1 | The Destination\_Unreachable property specifies a destination unreachable message. It should be generated by a router, or by the IPv6 later in the originating node, in response to a packet that cannot be delivered to its destination address for reasons other than congestion.  Only one of the Destination\_Unreachable, Packet\_Too\_Big, Time\_Exceeded and Parameter\_Problem properties can be populated. |
| **Packet\_Too\_Big** | ICMPv6PacketTooBigType | 0..1 | The Packet\_Too\_Big property specifies a packet too big message. It must be sent by a router in response to a packet that it cannot forward because the packet is larger than the MTU of the outgoing link.  Only one of the Destination\_Unreachable, Packet\_Too\_Big, Time\_Exceeded and Parameter\_Problem properties can be populated. |
| **Time\_Exceeded** | ICMPv6TimeExceededType | 0..1 | The Time\_Exceeded property specifies a time exceeded message. It is sent if either the hop limit is exceeded (hop limit = 0) or if fragment reassembly has timed out.  Only one of the Destination\_Unreachable, Packet\_Too\_Big, Time\_Exceeded and Parameter\_Problem properties can be populated. |
| **Parameter\_Problem** | ICMPv6ParameterProblemType | 0..1 | The Parameter\_Problem property specifies a parameter problem message. If an IPv6 node processing a packet finds a problem with a property in the IPv6 header or extension headers and it cannot complete processing the packet, it should send an ICMPv6 Parameter Problem message to the packet's source.  Only one of the Destination\_Unreachable, Packet\_Too\_Big, Time\_Exceeded and Parameter\_Problem properties can be populated. |

##### ICMPv6DestinationUnreachableType Class

The ICMPv6DestinationUnreachableType class specifies a destination unreachable error message; ICMP v6 class=1.

In CybOX 2.1.1, all of the properties of the ICMPv6DestinationUnreachableType class are mutually exclusive, i.e., only one property can be populated. This restriction is based on the fact each property is a Boolean value to indicate if the message is the one of seven types of Destination Unreachable messages. In future releases, this could be modelled as using one property for the subtype (code) and an enumeration.

The property table of the ICMPv6DestinationUnreachableType class is given in **Table 3‑65**.

Table 3‑65. Properties of the ICMPv6DestinationUnreachableType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **No\_Route** | basicDataTypes:Boolean | 0..1 | The No\_Route property indicates whether no route to destination exists (ICMP v6 code=0). |
| **Comm\_Prohibited** | basicDataTypes:Boolean | 0..1 | The Comm\_Prohibited property indicates whether communication with destination is administratively prohibited (ICMP v6 code=1). |
| **Beyond\_Scope** | basicDataTypes:Boolean | 0..1 | The Beyond\_Scope property indicates whether destination is beyond the scope of source address (ICMP v6 code =2). |
| **Address\_Unreachable** | basicDataTypes:Boolean | 0..1 | The Address\_Unreachable property indicates whether an address is unreachable (ICMP v6 code=3). |
| **Port\_Unreachable** | basicDataTypes:Boolean | 0..1 | The Port\_Unreachable property indicates whether a port is unreachable (ICMP v6 code=4). |
| **Src\_Addr\_Failed\_Policy** | basicDataTypes:Boolean | 0..1 | The Src\_Addr\_Failed\_Policy property indicates whether a source address failed the ingress/egress policy (ICMP v6 code=5). |
| **Reject\_Route** | basicDataTypes:Boolean | 0..1 | The Reject\_Route property indicates whether the route to destination should be rejected (ICMP v6 code=6). |

##### ICMPv6PacketTooBigType Class

The ICMPv6PacketTooBigType class specifies the packet too big error message; ICMP v6 class=2.

The property table of the ICMPv6PacketTooBigType class is given in **Table 3‑66**.

Table 3‑66. Properties of the ICMPv6PacketTooBigType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Packet\_Too\_Big** | basicDataTypes:Boolean | 0..1 | The Packet\_Too\_Big property is set to 0 (zero) by the originator and ignored by the receiver. |
| **MTU** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The MTU property specifies the Maximum Transmission Unit (MTU) size limit. |

##### ICMPv6TimeExceededType Class

The ICMPv6TimeExceededType class specifies the time exceeded error message; ICMP v6 class=3.

In CybOX 2.1.1, the properties, Hop\_Limit\_Exceeded and Fragment\_Reassem\_Time\_Exceeded are mutually exclusive, i.e., only one property can be populated. This restriction is based on that there are two different types of time exceeded messages. In future releases, it will probably be replaced with one property and a corresponding enumeration.

The property table of the ICMPv6TimeExceededType class is given in **Table 3‑67**.

Table 3‑67. Properties of the ICMPv6TimeExceededType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Hop\_Limit\_Exceeded** | basicDataTypes:Boolean | 0..1 | The Hop\_Limit\_Exceeded property specifies whether the hop limit was exceeded in transit (ICMP v6 code=0). |
| **Fragment\_Reassem\_Time\_Exceeded** | basicDataTypes:Boolean | 0..1 | The Fragment\_Reassem\_Time\_Exceeded property specifies whether the fragment reassembly time was exceeded (ICMP v6 code=1). |

##### ICMPv6ParameterProblemType Class

The ICMPv6ParameterProblemType class specifies a parameter problem error message; ICMP v6 class=4.

In CybOX 2.1.1, all of the properties of the ICMPv6ParameterProblemType class except Pointer are mutually exclusive, i.e., only one property can be populated. This restriction is based on the fact each property is a Boolean value to indicate if the message is the one of three types of Parameter Problem messages. In future releases, this could be modelled as using one property for the subtype (code) and an enumeration.

The property table of the ICMPv6ParameterProblemType class is given in **Table 3‑68**.

Table 3‑68. Properties of the ICMPv6ParameterProblemType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Pointer** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Pointer property specifies the octet offset within invoking packet where error was detected. |
| **Erroneous\_Header\_Field** | basicDataTypes:Boolean | 0..1 | The Erroneous\_Header\_Field property indicates whether an erroneous header field was encountered (ICMP v6 code=0).  Only one of the Erroneous\_Header\_Field, Unrecognized\_Next\_Header\_Type and Unrecognized\_IPv6\_Option properties can be populated. |
| **Unrecognized\_Next\_**  **Header\_Type** | basicDataTypes:Boolean | 0..1 | The Unrecognized\_Next\_Header\_Type property indicates whether an unrecognized next header type was encountered (ICMP v6 code=1).  Only one of the Erroneous\_Header\_Field, Unrecognized\_Next\_Header\_Type and Unrecognized\_IPv6\_Option properties can be populated. |
| **Unrecognized\_IPv6\_Option** | basicDataTypes:Boolean | 0..1 | The Unrecognized\_IPv6\_Option property indicates whether an unrecognized IP v6 option was encountered (ICMP v6 code=2).  Only one of the Erroneous\_Header\_Field, Unrecognized\_Next\_Header\_Type and Unrecognized\_IPv6\_Option properties can be populated. |

#### ICMPv6InfoMessageType Class

The ICMPv6InfoMessageType class specifies ICMP v6 informational messages include echo request/reply; other informational message class will be added in the future as they are more commonly used (only echo request/reply are defined in <http://tools.ietf.org/html/rfc4443>).

In CybOX 2.1.1, all of the properties of the ICMPv6InfoMessageType class, except Info\_Msg\_Content, are mutually exclusive, i.e., only one property can be populated. This restriction is based on the fact that the classes ICMPv6EchoReplyType, ICMPv6EchoRequestType, and any others added in the future can be modelled as subclasses of the ICMPv6InfoMessageType class. In future releases, it will probably be modelled that way.

The property table of the ICMPv6InfoMessageType class is given in **Table 3‑69**.

Table 3‑69. Properties of the ICMPv6InfoMessageType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Info\_Msg\_Content** | ICMPv6InfoMessageContentType | 0..1 | The Info\_Msg\_Content property specifies properties that are common to all ICMP v6 informational messages. Properties that are specific to individual messages are defined separately under each message type. |
| **Echo\_Request** | ICMPv6EchoRequestType | 0..1 | The Echo\_Request property specifies an echo request message (type=128). These messages are also known as "ping".  Only one of the Echo\_Reply and Echo\_Request properties can be populated. |
| **Echo\_Reply** | ICMPv6EchoReplyType | 0..1 | The Echo\_Reply property specifies an echo reply message (type=129). These messages are also known as "ping".  Only one of the Echo\_Reply and Echo\_Request properties can be populated. |

##### ICMPv6InfoMessageContentType Class

The ICMPv6InfoMessageContentType class specifies properties common to the ICMPv6 informational messages.

The property table of the ICMPv6InfoMessageContentType class is given in **Table 3‑70**.

Table 3‑70. Properties of the ICMPv6InfoMessageContentType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Identifier** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Identifier property specifies a 16-bit identifier, which is combined with the sequence number, and is called the "quench" for echo reply and echo request. |
| **Sequence\_Number** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Sequence\_Number property specifies a 16-bit sequence number. The identifier and sequence number can be used by the client to match the reply with the request that caused the reply. |

##### ICMPv6EchoRequestType Class

The ICMPv6EchoRequestType class specifies an echo request informational ICMP v6 message; class=128.

The property table of the ICMPv6EchoRequestType class is given in **Table 3‑71**.

Table 3‑71. Properties of the ICMPv6EchoRequestType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Echo\_Request** | basicDataTypes:Boolean | 0..1 | The Echo\_Request property specifies whether the subtype of the message is an echo request message (code=128). |
| **Data** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Data property specifies zero or more octets of arbitrary data. |

##### ICMPv6EchoReplyType Class

Echo reply informational ICMP v6 message; class=129.

The property table of the ICMPv6EchoReplyType class is given in **Table 3‑72**.

Table 3‑72. Properties of the ICMPv6EchoReplyType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Echo\_Reply** | basicDataTypes:Boolean | 0..1 | The Echo\_Reply property specifies whether the subtype of the message is an echo reply message (code=129). |
| **Data** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Data property specifies the data resulting from the invoking echo request message. |

## TransportLayerType Class

The TransportLayerType class specifies the properties of the UDP and TCP protocol. Other protocols will be defined as necessary.

In CybOX 2.1.1, the TCP and UDP properties are mutually exclusive, i.e., only one property can be populated. This restriction is based on the fact that the classes TCPType and UDPType can be modelled as subclasses of the TransportLayerType class. In future releases, it will probably be modelled that way.

The property table of the TransportLayerType class is given in **Table 3‑73**.

Table 3‑73. Properties of the TransportLayerType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **TCP** | TCPType | 0..1 | The TCP property specifies a TCP packet. TCP provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer.  Only one of the TCP and UDP properties can be populated. |
| **UDP** | UDPType | 0..1 | The UDP property specifies a UDP packet. UDP uses a simple transmission model without implicit handshaking dialogues for providing reliability, ordering, or data integrity. Thus, UDP provides an unreliable service and datagrams may arrive out of order, appear duplicated, or go missing without notice.  Only one of the TCP and UDP properties can be populated. |

### TCPType Class

The TCPType class specifies a TCP packet. The TCP protocol provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer. See <http://en.wikipedia.org/wiki/Transmission_Control_Protocol> for more information.

The UML diagram corresponding to the TCPType class is shown in **Figure 3‑6**.

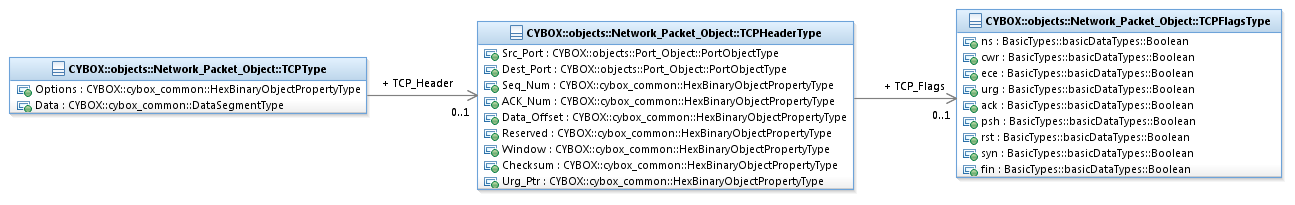


Figure 3‑6. UML diagram of the TCPType class

The property table of the TCPType class is given in **Table 3‑74**.

Table 3‑74. Properties of the TCPType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **TCP\_Header** | TCPHeaderType | 0..1 | The TCP\_Header property specifies the 10 mandatory properties and an optional extension property. |
| **Options** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Option property contains the TCP Options. There are up to three properties: Option-Kind (1 byte), Option-Length (1 byte), Option-Data (variable). This property will be further defined when required. |
| **Data** | cyboxCommon:DataSegmentType | 0..1 | The Data property specifies the data payload of the TCP packet. |

#### TCPHeaderType Class

The TCPHeaderType class contains 10 mandatory fields and an optional extension field.

The property table of the TCPHeaderType class is given in **Table 3‑75**.

Table 3‑75. Properties of the TCPHeaderType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Src\_Port** | PortObj:PortObjectType | 0..1 | The Src\_Port property specifies the sending port. |
| **Dest\_Port** | PortObj:PortObjectType | 0..1 | The Dest\_Port property specifies the receiving port. |
| **Seq\_Num** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Seq\_Num property specifies the Sequence number (32-bits). The Sequence number has a dual role: If the syn flag is set, then this is the initial sequence numbers. If the syn flag is clear (see TCP\_Flags property), then this is the accumulated sequence number of the first data byte of this packet for the current session. |
| **ACK\_Num** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The ACK\_Num property specifies whether the ack flag (see TCP\_Flags property) is set then the value of this property is the next sequence number that the receiver is expecting. |
| **Data\_Offset** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Data\_Offset property specifies the size of the TCP header in 32-bit words. |
| **Reserved** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Reserved property specifies that these 3 bits are reserved for future use and should be set to zero. |
| **TCP\_Flags** | TCPFlagsType | 0..1 | The TCP\_Flags property specifies the 9 flags (aka Control Bits). |
| **Window** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Window property specifies the size of the receive window, which specifies the number of bytes (beyond the sequence number in the acknowledgment property) that the sender of this segment is currently willing to receive. |
| **Checksum** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Checksum property specifies the 16-bit checksum which is used for error-checking of the header and data. |
| **Urg\_Ptr** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Urg\_Ptr property specifies a 16-bit property that is an offset from the sequence number indicating the last urgent data byte when the urg flag is set |

#### TCPFlagsType Class

The TCPFlagsType class specifies the nine different flags in the TCP header.

The property table of the TCPFlagsType class is given in **Table 3‑76**.

Table 3‑76. Properties of the TCPFlagsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **ns** | basicDataTypes:Boolean | 0..1 | The ns property specifies the ECN-nonce concealment protection setting. |
| **cwr** | basicDataTypes:Boolean | 0..1 | The cwr property specifies whether the Congestion Window Reduced (CWR) flag is set by the sending host to indicate that it received a TCP segment with the ECE flag set and had responded in congestion control mechanism. |
| **ece** | basicDataTypes:Boolean | 0..1 | The ece property specifies the ECN-Echo flag has a dual role: if the syn flag is set, the TCP peer is ECN capable; if the syn flag is clear, a packet with Congestion Experienced flag set (ECN=11) in IP header is received during normal transmission (see  [http://tools.ietf.org/html/rfc3168](%20http://tools.ietf.org/html/rfc3168)). |
| **urg** | basicDataTypes:Boolean | 0..1 | The urg property specifies whether the Urg\_Ptr property is significant. |
| **ack** | basicDataTypes:Boolean | 0..1 | The ack property specifies whether the Acknowledgment property is significant. All packets after the initial syn packet sent by the client should have this flag set. |
| **psh** | basicDataTypes:Boolean | 0..1 | The psh property specifies whether to push the buffered data to the receiving application. |
| **rst** | basicDataTypes:Boolean | 0..1 | The rst property specifies whether to reset the connection. |
| **syn** | basicDataTypes:Boolean | 0..1 | The syn property specifies whether to synchronize sequence numbers. Only the first packet sent from each end should have this flag set. |
| **fin** | basicDataTypes:Boolean | 0..1 | The fin property specifies whether there is no more data from sender. |

### UDPType Class

The UDPType class specifies a UDP packet. The UDP protocol uses a simple transmission model without implicit handshaking dialogues for providing reliability, ordering, or data integrity. Thus, UDP provides an unreliable service and datagrams may arrive out of order, appear duplicated, or go missing without notice. See <http://en.wikipedia.org/wiki/User_Datagram_Protocol> for more information.

The UML diagram corresponding to the UDPType class is shown in **Figure 3‑7**.

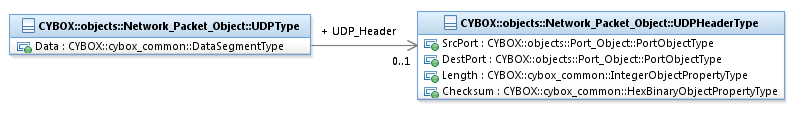


Figure 3‑7. UML diagram for UDPType class

The property table of the UDPType class is given in **Table 3‑77**.

Table 3‑77. Properties of the UDPType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **UDP\_Header** | UDPHeaderType | 0..1 | The UDP\_Header properties consists of four properties. |
| **Data** | cyboxCommon:DataSegmentType | 0..1 | The Data property specifies the data payload of the UDP packet. |

#### UDPHeaderType Class

The UDP header class defines the four fields in the UDP header.

The property table of the UDPHeaderType class is given in **Table 3‑78**.

Table 3‑78. Properties of the UDPHeaderType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **SrcPort** | PortObj:PortObjectType | 0..1 | The SrcPort property specifies the sender's port. |
| **DestPort** | PortObj:PortObjectType | 0..1 | The DestPort property specifies the receiver's port. |
| **Length** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Length property specifies the length in bytes of the entire datagram (header and data). |
| **Checksum** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Checksum property specifies the checksum is used for error-checking of the header and data. |

## IANAPortNumberRegistryType Data Type[[1]](#endnote-1)

The IANAPortNumberRegistryType data type specifies the port numbers. Its core value SHOULD be a literal found in the IANAPortNumberRegistryTypeEnum enumeration. Its base type is the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

## Enumerations

### ARPOpTypeEnum Enumeration

The literals of the ARPOpTypeEnum enumeration are given in **Table 3‑79**.

Table 3‑79. Literals of the ARPOpTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **ARP request(1)** | Indicates the ARP request operation, or value 1 in the OPER field of an ARP packet. |
| **ARP reply(2)** | Indicates the ARP reply operation, or value 2 in the OPER field of an ARP packet. |
| **RARP request(3)** | Indicates the RARP request operation, or value 3 in the OPER field of an ARP packet. |
| **RARP reply(4)** | Indicates the RARP reply operation, or value 4 in the OPER field of an ARP packet. |

### DoNotFragmentTypeEnum Enumeration

The literals of the DoNotFragmentTypeEnum enumeration are given in **Table 3‑80**.

Table 3‑80. Literals of the DoNotFragmentTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **fragementifnecessary(0)** | Indicates that the router or other device should fragment the packet if necessary, especially if the packet size is bigger than the MTU of an outgoing interface. |
| **donotfragment(1)** | Indicates that the router or other device should NOT fragment the packet in any circumstance. |

### MoreFragmentsTypeEnum Enumeration

The literals of the MoreFragmentsTypeEnum enumeration are given in **Table 3‑81**.

Table 3‑81. Literals of the MoreFragmentsTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **lastfragment(0)** | Indicates that the last fragment has been received. In other words, the "more fragments" flag is set to 0. |
| **morefragmentstofollow(1)** | Indicates that more fragments need to be received. In other words, the "more fragments" flag is set. |

### IPv4CopyFlagTypeEnum Enumeration

The literals of the IPv4CopyFlagTypeEnum enumeration are given in **Table 3‑82**.

Table 3‑82. Literals of the IPv4CopyFlagTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **donotcopy(0)** | Indicates that the options need NOT be copied into all fragments of a fragmented packet. |
| **copy(1)** | Indicates that the options need to be copied into all fragments of a fragmented packet. |

### IPv4ClassTypeEnum Enumeration

The literals of the IPv4ClassTypeEnum enumeration are given in **Table 3‑83**.

Table 3‑83. Literals of the IPv4ClassTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **control(0)** | Indicates the "control" options. |
| **reserved(1)** | Indicates a reserved value. |
| **debuggingandmeasurement(2)** | Indicates the debugging and measurement options. |
| **reserved(3)** | Indicates a reserved value. |

### IPv4OptionsTypeEnum Enumeration

The literals of the IPv4OptionsTypeEnum enumeration are given in **Table 3‑84**.

Table 3‑84. Literals of the IPv4OptionsTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **endofoptionslist(0)** | Indicates the End of Options List option, or EOOL. |
| **nop(1)** | Indicates the No Operation option, or NOP. |
| **security(2)** | Indicates the Security option, or SEC. |
| **loosesourceroute(3)** | Indicates the Loose Source Route option, or LSR. |
| **timestamp(4)** | Indicates the Time Stamp option, or TS. |
| **extendedsecurity(5)** | Indicates the Extended Security option, or E-SEC. |
| **commercialsecurity(6)** | Indicates the Commercial Security option, or CIPSO. |
| **recordroute(7)** | Indicates the Record Route option, or RR. |
| **streamidentifier(8)** | Indicates the Stream ID option, or SID. |
| **strictsourceroute(9)** | Indicates the Strict Source Route option, or SSR. |
| **experimentalmeasure(10)** | Indicates the Experimental Measurement option, or ZSU. |
| **mtuprobe(11)** | Indicates the MTU probe option, or MTUP. |
| **mtureply(12)** | Indicates the MTU reply option, or MTUR. |
| **experimentalflowcontrol(13)** | Indicates the Experimental Flow Control option, or FINN. |
| **experimentalaccesscontrol(14)** | Indicates the Experimental Access Control option, or FINN. |
| **encode(15)** |  |
| **imitrafficdescriptor(16)** | Indicates the IMI Traffic Descriptor option, or IMITD. |
| **extendedip(17)** | Indicates the Extended Internet Protocol option, or EIP. |
| **traceroute(18)** | Indicates the Trace Route option, or TR. |
| **addressextension(19)** | Indicates the Address Extension option, or ADDEXT. |
| **routeralert(20)** | Indicates a Router Alert option, or RTRALT. |
| **selectivedirectedbroadcasemode(21)** | Indicates a Selective Directed Broadcast option, or SDB. |
| **dynamicepacketstate(23)** | Indicates the Dynamic Packet State option, or DPS. |
| **upstreammulticastpacket(24)** | Indicates the Upstream Multicast Packet option, or UMP. |
| **quickstart(25)** | Indicates the Quick-Start option, or QS. |
| **exp(30)** | Indicates the RFC3692-style Experiment option, or EXP. |

### IPv6DoNotRecogActionTypeEnum Enumeration

The literals of the IPv6DoNotRecogActionTypeEnum enumeration are given in **Table 3‑85**.

See <http://tools.ietf.org/html/rfc2460> for more information.

Table 3‑85. Literals of the IPv6DoNotRecogActionTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **skipoption(00)** | Indicates that the option should be skipped and the header should continue to be processed. |
| **discardpacket(01)** | Indicates that the packet should be discarded. |
| **discardpacketsendicmpcode2(10)** | Indicates that the packet should be discarded and regardless of whether or not the packet's Destination Address was a multicast address, send an ICMP Parameter Problem, Code 2, message to the packet's Source Address, pointing to the unrecognized Option Type. |
| **discardpacketsendicmpcode2nomulti(11)** | Indicates that the packet should be discarded and only if the packet's Destination Address was not a multicast address, send an ICMP Parameter Problem, Code 2, message to the packet's Source Address, pointing to the unrecognized Option |

### IPv6PacketChangeTypeEnum Enumeration

The literals of the IPv6PacketChangeTypeEnum enumeration are given in **Table 3‑86**.

See <http://tools.ietf.org/html/rfc2460> for more information.

Table 3‑86. Literals of the IPv6PacketChangeTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **nochange(0)** | Indicates that the packet does not change en-route. |
| **change(1)** | Indicates that the packet may change en-route. |

### IPVersionTypeEnum Enumeration

The literals of the IPVersionTypeEnum enumeration are given in **Table 3‑87**.

Table 3‑87. Literals of the IPVersionTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **IPv4(4)** | Indicates IP Version 4. |
| **ST(5)** | Indicates the IP version designating ST Datagram Mode. |
| **IPv6(6)** | Indicates IP Version 6. |
| **TP/IX(7)** | Indicates the IP version designating TP/IX: The Next Internet. |
| **PIP(8)** | Indicates the IP version designating PIP: The P Internet Protocol. |
| **TUBA(9)** | Indicates the IP version designating TUBA (TCP and UDP with Bigger Addresses, i.e. <http://tools.ietf.org/html/rfc1347>). |

### IANAHardwareTypeEnum Enumeration

The literals of the IANAHardwareTypeEnum enumeration are given in **Table 3‑88**.

Table 3‑88. Literals of the IANAHardwareTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **Ethernet(1)** | Indicates Ethernet hardware. |
| **IEEE802(6)** | Indicates IEEE 802 compliant hardware for networks carrying variable-size packets. |
| **ARCNET(7)** | Indicates the ARCNET LAN protocol. |
| **FrameRelay(15)** | Indicates the Frame Relay WAN technology. |
| **ATM(16)** | Indicates the ATM (Asynchronous Transfer Mode) networking standard. |
| **HDLC(17)** | Indicates the HDLC (High-Level Data Link Control) protocol. |
| **FibreChannel(18)** | Indicates the FibreChannel technology. |
| **ATM(19)** | Indicates the ATM (Asynchronous Transfer Mode) networking standard. |
| **SerialLine(20)** | Indicates the Serial Line protocol, or SLIP. |

### IANAEtherTypeEnum Enumeration

The literals of the IANAEtherTypeEnum enumeration are given in **Table 3‑89**.

Table 3‑89. Literals of the IANAEtherTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **IPv4(0x0800)** | Indicates the IPv4 Ethernet type is specified. |
| **ARP(0x0806)** | Indicates the ARP Ethernet type is specified. |
| **RARP(0x8035)** | Indicates the RARP Ethernet type is specified. |
| **IPX(0x8137)** | Indicates the IPX Ethernet type is specified. |
| **SNMP(0x814C)** | Indicates the SNMP Ethernet type is specified. |
| **IPv6(0x86DD)** | Indicates the IPv6 Ethernet type is specified. |

### IANAAssignedIPNumbersTypeEnum Enumeration

The literals of the IANAAssignedIPNumbersTypeEnum enumeration are given in **Table 3‑90**.

Table 3‑90. Literals of the IANAAssignedIPNumbersTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **IPv6hopbyhop(0)** | Indicates the IPv6 Hop-By-Hop option protocol (HOPOPT). |
| **ICMP(1)** | Indicates the Internet Control Message protocol (HOPOPT). |
| **IGMP(2)** | Indicates the Internet Group Message protocol (HOPOPT). |
| **GGP(3)** | Indicates the Gateway-to-Gateway protocol (HOPOPT). |
| **IPv4Encapsulation(4)** | Indicates the IPv4 Encapsulation protocol (IPv4). |
| **ST(5)** | Indicates the Stream protocol (HOPOPT). |
| **TCP(6)** | Indicates the TCP protocol. |
| **EGP(8)** | Indicates the EGP (Exterior Gateway) protocol. |
| **IGRP(9)** | Indicates the IGP/IGRP (Cisco) protocol. |
| **NVP(11)** | Indicates the Network-Voice protocol. |
| **PUP(12)** | Indicates the PUP protocol. |
| **ARGUS(13)** | Indicates the ARGUS protocol. |
| **EMCON(14)** | Indicates the EMCON protocol. |
| **XNET(15)** | Indicates the Cross Net Debugger protocol. |
| **UDP(17)** | Indicates the UDP protocol. |
| **IPv6Encapsulation(41)** | Indicates the IPv6 protocol. |
| **SDRP(42)** | Indicates the Source Demand Routing protocol. |
| **IPv6routingheader(43)** | Indicates the routing header for IPv6. |
| **IPv6fragmentheader(44)** | Indicates the fragment header for IPv6. |
| **RSVP(46)** | Indicates the Reservation Protocol. |
| **GRE(47)** | Indicates the General Routing Encapsulation protocol number. |
| **encapsultaesecuritypayload\_ESP(50)** | Indicates the Encapsulated Security Payload protocol number. |
| **authenticationheader\_AH(51)** | Indicates the Authentication Header protocol number. |
| **ICMPv6(58)** | Indicates the ICMP for v6 protocol number. |
| **IPv6nonextheader(59)** | Indicates the No Next Header for IPv6 protocol number. |
| **IPv6destinationoptions(60)** | Indicates the Destination Options for IPv6 protocol number. |
| **mobilityheader(135)** | Indicates the Mobility Header protocol number. |

### IANAPortNumberRegistryTypeEnum Enumeration

The literals of the IANAPortNumberRegistryTypeEnum enumeration are given in **Table 3‑91**.

Table 3‑91. Literals of the IANAPortNumberRegistryTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **ftpdata(20)** | Indicates the port for ftpdata. |
| **ftp(21)** | Indicates the port for ftp. |
| **ssh(22)** | Indicates the port for ssh. |
| **telnet(23)** | Indicates the port for telnet. |
| **smtp(25)** | Indicates the port for smtp. |
| **domain(53)** | Indicates the domain port. |
| **tftp(69)** | Indicates the port for tftp. |
| **http(80)** | Indicates the port for http. |
| **ldap(389)** | Indicates the port for ldap. |
| **https(443)** | Indicates the port for https. |

### MFlagTypeEnum Enumeration

The literals of the MFlagTypeEnum enumeration are given in **Table 3‑92**.

Table 3‑92. Literals of the MFlagTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **lastfragment(0)** | Fragment is the last fragment. |
| **morefragments(1)** | There are more fragments (current is not the last). |

# Conformance

Implementations have discretion over which parts (components, properties, extensions, controlled vocabularies, etc.) of CybOX they implement (e.g., Observable/Object).

[1] Conformant implementations must conform to all normative structural specifications of the UML model or additional normative statements within this document that apply to the portions of CybOX they implement (e.g., implementers of the entire Observable class must conform to all normative structural specifications of the UML model regarding the Observable class or additional normative statements contained in the document that describes the Observable class).

[2] Conformant implementations are free to ignore normative structural specifications of the UML model or additional normative statements within this document that do not apply to the portions of CybOX they implement (e.g., non-implementers of any particular properties of the Observable class are free to ignore all normative structural specifications of the UML model regarding those properties of the Observable class or additional normative statements contained in the document that describes the Observable class).

The conformance section of this document is intentionally broad and attempts to reiterate what already exists in this document.

1. Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

|  |  |
| --- | --- |
| **Aetna**  David Crawford  **AIT Austrian Institute of Technology**  Roman Fiedler  Florian Skopik  **Australia and New Zealand Banking Group (ANZ Bank)**  Dean Thompson  **Blue Coat Systems, Inc.**  Owen Johnson  Bret Jordan  **Century Link**  Cory Kennedy  **CIRCL**  Alexandre Dulaunoy  Andras Iklody  Raphaël Vinot  **Citrix Systems**  Joey Peloquin  **Dell**  Will Urbanski  Jeff Williams  **DTCC**  Dan Brown  Gordon Hundley  Chris Koutras  **EMC**  Robert Griffin  Jeff Odom  Ravi Sharda  **Financial Services Information Sharing and Analysis Center (FS-ISAC)**  David Eilken  Chris Ricard  **Fortinet Inc.**  Gavin Chow  Kenichi Terashita  **Fujitsu Limited**  Neil Edwards  Frederick Hirsch  Ryusuke Masuoka  Daisuke Murabayashi  **Google Inc.**  Mark Risher  **Hitachi, Ltd.**  Kazuo Noguchi  Akihito Sawada  Masato Terada  **iboss, Inc**.  Paul Martini  **Individual**  Jerome Athias  Peter Brown  Elysa Jones  Sanjiv Kalkar  Bar Lockwood  Terry MacDonald  Alex Pinto  **Intel Corporation**  Tim Casey  Kent Landfield  **JPMorgan Chase Bank, N.A.**  Terrence Driscoll  David Laurance  **LookingGlass**  Allan Thomson  Lee Vorthman  **Mitre Corporation**  Greg Back  Jonathan Baker  Sean Barnum  Desiree Beck  Nicole Gong  Jasen Jacobsen  Ivan Kirillov  Richard Piazza  Jon Salwen  Charles Schmidt  Emmanuelle Vargas-Gonzalez  John Wunder  **National Council of ISACs (NCI)**  Scott Algeier  Denise Anderson  Josh Poster  **NEC Corporation**  Takahiro Kakumaru  **North American Energy Standards Board**  David Darnell  **Object Management Group**  Cory Casanave  **Palo Alto Networks**  Vishaal Hariprasad  **Queralt, Inc**.  John Tolbert  **Resilient Systems, Inc.**  Ted Julian  **Securonix**  Igor Baikalov  **Siemens AG**  Bernd Grobauer  **Soltra**  John Anderson  Aishwarya Asok Kumar  Peter Ayasse  Jeff Beekman  Michael Butt  Cynthia Camacho  Aharon Chernin  Mark Clancy  Brady Cotton  Trey Darley  Mark Davidson  Paul Dion  Daniel Dye  Robert Hutto  Raymond Keckler  Ali Khan  Chris Kiehl  Clayton Long  Michael Pepin  Natalie Suarez  David Waters  Benjamin Yates  **Symantec Corp.**  Curtis Kostrosky  **The Boeing Company**  Crystal Hayes  **ThreatQuotient, Inc.**  Ryan Trost  **U.S. Bank**  Mark Angel  Brad Butts  Brian Fay  Mona Magathan  Yevgen Sautin  **US Department of Defense (DoD)**  James Bohling  Eoghan Casey  Gary Katz  Jeffrey Mates  **VeriSign**  Robert Coderre  Kyle Maxwell  Eric Osterweil | **Airbus Group SAS**  Joerg Eschweiler  Marcos Orallo  **Anomali**  Ryan Clough  Wei Huang  Hugh Njemanze  Katie Pelusi  Aaron Shelmire  Jason Trost  **Bank of America**  Alexander Foley  **Center for Internet Security (CIS)**  Sarah Kelley  **Check Point Software Technologies**  Ron Davidson  **Cisco Systems**  Syam Appala  Ted Bedwell  David McGrew  Pavan Reddy  Omar Santos  Jyoti Verma  **Cyber Threat Intelligence Network, Inc. (CTIN)**  Doug DePeppe  Jane Ginn  Ben Othman  **DHS Office of Cybersecurity and Communications (CS&C)**  Richard Struse  Marlon Taylor  **EclecticIQ**  Marko Dragoljevic  Joep Gommers  Sergey Polzunov  Rutger Prins  Andrei Sîrghi  Raymon van der Velde  **eSentire, Inc.**  Jacob Gajek  **FireEye, Inc.**  Phillip Boles  Pavan Gorakav  Anuj Kumar  Shyamal Pandya  Paul Patrick  Scott Shreve  **Fox-IT**  Sarah Brown  **Georgetown University**  Eric Burger  **Hewlett Packard Enterprise (HPE)**  Tomas Sander  **IBM**  Peter Allor  Eldan Ben-Haim  Sandra Hernandez  Jason Keirstead  John Morris  Laura Rusu  Ron Williams  **IID**  Chris Richardson  **Integrated Networking Technologies, Inc.**  Patrick Maroney  **Johns Hopkins University Applied Physics Laboratory**  Karin Marr  Julie Modlin  Mark Moss  Pamela Smith  **Kaiser Permanente**  Russell Culpepper  Beth Pumo  **Lumeta Corporation**  Brandon Hoffman  **MTG Management Consultants, LLC.**  James Cabral  **National Security Agency**  Mike Boyle  Jessica Fitzgerald-McKay  **New Context Services, Inc.**  John-Mark Gurney  Christian Hunt  James Moler  Daniel Riedel  Andrew Storms  **OASIS**  James Bryce Clark  Robin Cover  Chet Ensign  **Open Identity Exchange**  Don Thibeau  **PhishMe Inc.**  Josh Larkins  **Raytheon Company-SAS**  Daniel Wyschogrod  **Retail Cyber Intelligence Sharing Center (R-CISC)**  Brian Engle  **Semper Fortis Solutions**  Joseph Brand  **Splunk Inc.**  Cedric LeRoux  Brian Luger  Kathy Wang  **TELUS**  Greg Reaume  Alan Steer  **Threat Intelligence Pty Ltd**  Tyron Miller  Andrew van der Stock  **ThreatConnect, Inc.**  Wade Baker  Cole Iliff  Andrew Pendergast  Ben Schmoker  Jason Spies  **TruSTAR Technology**  Chris Roblee  **United Kingdom Cabinet Office**  Iain Brown  Adam Cooper  Mike McLellan  Chris O’Brien  James Penman  Howard Staple  Chris Taylor  Laurie Thomson  Alastair Treharne  Julian White  Bethany Yates  **US Department of Homeland Security**  Evette Maynard-Noel  Justin Stekervetz  **ViaSat, Inc.**  Lee Chieffalo  Wilson Figueroa  Andrew May  **Yaana Technologies, LLC**  Anthony Rutkowski |

The authors would also like to thank the larger CybOX Community for its input and help in reviewing this document.

1. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Editor** | **Changes Made** |
| wd01 | 15 December 2015 | Desiree Beck Trey Darley Ivan Kirillov Rich Piazza | Initial transfer to OASIS template |

1. This class is not used in this data model. It is used in the Network Flow data model and should be moved there in a future release. [↑](#endnote-ref-1)