CybOXTM Version 2.1.1 Part 69: Win Executable File 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*. [URI]
* *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*. (this document)
* *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 Win Executable File 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 2015. 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 2012-2015.  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 7](#_Toc438562610)

[1.1 CybOXTM Specification Documents 7](#_Toc438562611)

[1.2 Document Conventions 7](#_Toc438562612)

[1.2.1 Fonts 7](#_Toc438562613)

[1.2.2 UML Package References 8](#_Toc438562614)

[1.2.3 UML Diagrams 8](#_Toc438562615)

[1.2.3.1 Class Properties 8](#_Toc438562616)

[1.2.3.2 Diagram Icons and Arrow Types 8](#_Toc438562617)

[1.2.4 Property Table Notation 9](#_Toc438562618)

[1.2.5 Property and Class Descriptions 9](#_Toc438562619)

[1.3 Terminology 10](#_Toc438562620)

[1.4 Normative References 10](#_Toc438562621)

[2 Background Information 11](#_Toc438562622)

[2.1 Cyber Observables 11](#_Toc438562623)

[2.2 Objects 11](#_Toc438562624)

[3 Data Model 12](#_Toc438562625)

[3.1 WindowsExecutableFileObjectType Class 12](#_Toc438562626)

[3.2 PEChecksumType Class 14](#_Toc438562627)

[3.3 PEExportsType Class 15](#_Toc438562628)

[3.4 PEExportedFunctionsType Class 16](#_Toc438562629)

[3.5 PESectionListType Class 16](#_Toc438562630)

[3.6 EntropyType Class 17](#_Toc438562631)

[3.7 PEImportType Class 17](#_Toc438562632)

[3.8 PEImportedFunctionsType Class 18](#_Toc438562633)

[3.9 PEResourceType Class 18](#_Toc438562634)

[3.10 PEVersionInfoResourceType Class 19](#_Toc438562635)

[3.11 PEExportedFunctionType Class 21](#_Toc438562636)

[3.12 PEResourceListType Class 21](#_Toc438562637)

[3.13 PEImportedFunctionType Class 22](#_Toc438562638)

[3.14 PEImportListType Class 23](#_Toc438562639)

[3.15 PESectionType Class 24](#_Toc438562640)

[3.16 PEDataDirectoryStructType Class 25](#_Toc438562641)

[3.17 PESectionHeaderStructType Class 25](#_Toc438562642)

[3.18 DOSHeaderType Class 27](#_Toc438562643)

[3.19 PEHeadersType Class 30](#_Toc438562644)

[3.20 PEFileHeaderType Class 32](#_Toc438562645)

[3.21 SubsystemType Data Type 33](#_Toc438562646)

[3.22 PEType Data Type 33](#_Toc438562647)

[3.23 PEOptionalHeaderType Class 33](#_Toc438562648)

[3.24 DataDirectoryType Class 37](#_Toc438562649)

[3.25 PEBuildInformationType Class 38](#_Toc438562650)

[3.26 PEResourceContentType Data Type 39](#_Toc438562651)

[3.27 SubsystemTypeEnum Enumeration 39](#_Toc438562652)

[3.28 PETypeEnum Enumeration 40](#_Toc438562653)

[3.29 PEResourceTypeEnum Enumeration 40](#_Toc438562654)

[4 Conformance 43](#_Toc438562655)

[Acknowledgments 44](#_Toc438562656)

[Revision History 45](#_Toc438562657)

# 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 Win Executable File 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 Win Executable File Object data model. We present the Win Executable File 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*](file:///C:\Users\drothenberg\Documents\GitHub\cyboxproject\specifications\documents\cybox-v2.1.1-wd01-part43-pdf-file-object.docx#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 package\_prefix for the Windows Executable File data model is WinExecutableFileObj. Note that in this specification document, we do not explicitly specify the package prefix for any classes that originate from the Win Executable File 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 Win Executable File 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 Win Executable File 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

Objects in CybOX are individual data models for characterizing 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.

# Data Model

## WindowsExecutableFileObjectType Class

The WindowsExecutableFileObjectType class is intended to characterize Windows PE (Portable Executable) files. The UML diagram corresponding to the WindowsExecutableFileObjectType class is shown in **Figure 3‑1**.

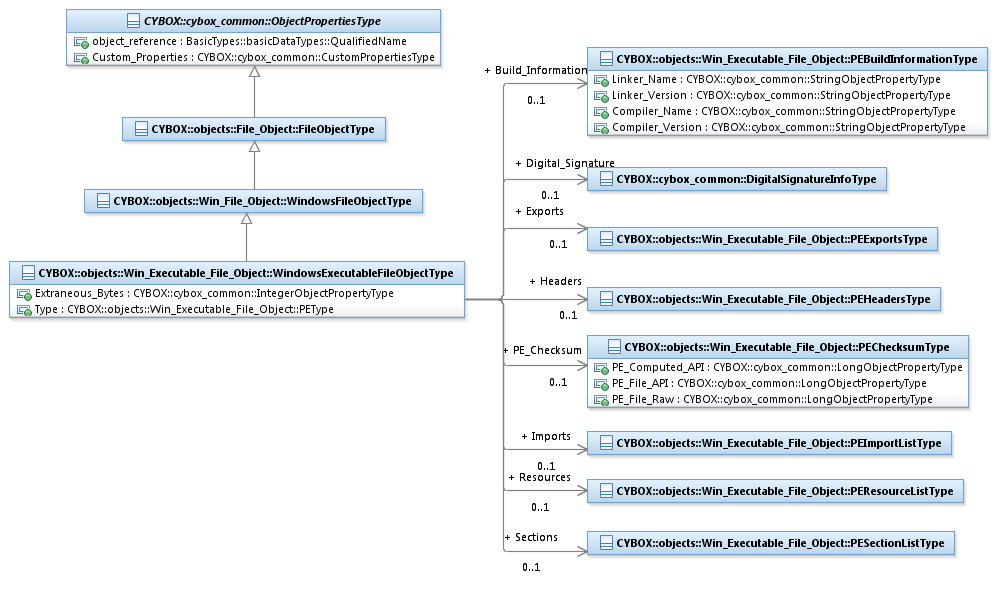


Figure 3‑1. UML diagram of the WindowsExecutableFileObjectType class

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

Table 3‑1. Properties of the WindowsExecutableFileObjectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Build\_Information** | PEBuildInformationType | 0..1 | The Build\_Information property specifies some information on the tools used to build the PE binary. |
| **Digital\_Signature** | cyboxCommon:  DigitalSignatureInfoType | 0..1 | The Digital\_Signature property specifies the information about the digital signature used to sign the PE binary. |
| **Exports** | PEExportsType | 0..1 | The Exports property characterizes the PE Export table of the PE Binary. |
| **Extraneous\_Bytes** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Extraneous\_Bytes property specifies the number of extraneous bytes contained in the PE binary. |
| **Headers** | PEHeadersType | 0..1 | The Headers property contains fields for characterizing aspects the various types of PE headers. |
| **Imports** | PEImportListType | 0..1 | The Imports property characterizes the PE Import Table of the binary. |
| **PE\_Checksum** | PEChecksumType | 0..1 | The PE\_Checksum property specifies the checksum of the PE file. |
| **Resources** | PEResourceListType | 0..1 | The Resources property characterizes the PE Resources of the binary. |
| **Sections** | PESectionListType | 0..1 | The Sections property characterizes the PE Sections of the binary. |
| **Type** | PEType | 0..1 | The Type property specifies the particular type of the PE binary, e.g. Executable. |

## PEChecksumType Class

The PEChecksumType class records the checksum of the PE file, both as found in the file and computed.

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

Table 3‑2. Properties of the PEChecksumType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **PE\_Computed\_API** | cyboxCommon:  LongObjectPropertyType | 0..1 | The PE\_Computed\_API property specifies a checksum computed by an external algorithm. |
| **PE\_File\_API** | cyboxCommon:  LongObjectPropertyType | 0..1 | The PE\_File\_API property specified the checksum computed by IMAGHELP.DLL. |
| **PE\_File\_Raw** | cyboxCommon:  LongObjectPropertyType | 0..1 | The PE\_File\_Raw property specifies the checksum found in the PE file (in the Optional Header). |

## PEExportsType Class

The PEExportsType class specifies the PE File exports data section. The exports data section contains information about symbols exported by the PE File (a DLL) which can be dynamically loaded by other executables. This class and its components abstract the Windows structures. The UML diagram corresponding to the PEExportsType class is shown in **Figure 3‑2**.

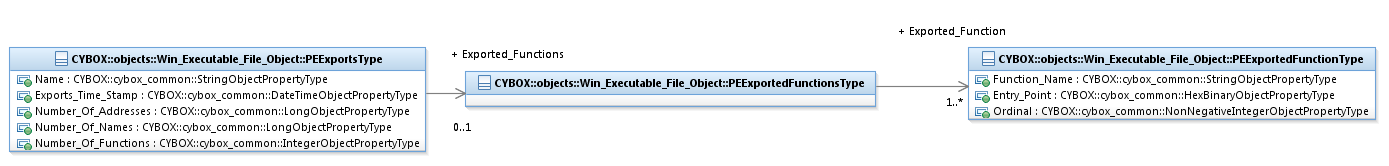


Figure 3‑2. UML diagram for PEExportsType class

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

Table 3‑3. Properties of the PEExportsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Name** | cyboxCommon:  StringObjectPropertyType | 0..1 | The Name property specifies the actual name of the PE module, as used by the PE loader when it is imported by another executable. |
| **Exported\_Functions** | PEExportedFunctionsType | 0..1 | The Exported\_Functions property specifies a list of the exported functions in this section. |
| **Exports\_Time\_Stamp** | cyboxCommon:  DateTimeObjectPropertyType | 0..1 | The Exports\_Time\_Stamp property specifies the date and time the export data was created. |
| **Number\_Of\_Addresses** | cyboxCommon:  LongObjectPropertyType | 0..1 | The Number\_Of\_Addresses property specifies the number of addresses in the export data section's address table. |
| **Number\_Of\_Names** | cyboxCommon:  LongObjectPropertyType | 0..1 | The Number\_Of\_Names property specifies the number of names in the export data section's name table. |
| **Number\_Of\_Functions** | cyboxCommon:  IntegerObjectPropertyType | 0..1 | The Number\_Of\_Functions property specifies the total number of functions that are exported by the PE file. |

## PEExportedFunctionsType Class

The PEExportedFunctionsType class specifies a list of PE exported functions.

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

Table 3‑4. Properties of the PEExportedFunctionsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Exported\_Function** | PEExportedFunctionType | 1..\* | TheExported\_Function property specifies a single property in the list of exported functions. |

## PESectionListType Class

The PESectionListType class captures a list of sections that appear in the PE file.

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

Table 3‑5. Properties of the PESectionListType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Section** | PESectionType | 1..\* | Specifies a property of a list of PE file sections. |

## EntropyType Class

The EntropyType class captures the result of an entropy computation.

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

Table 3‑6. Properties of the EntropyType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Value** | cyboxCommon:  FloatObjectPropertyType | 0..1 | The Value property specifies the computed entropy value. |
| **Min** | cyboxCommon:  FloatObjectPropertyType | 0..1 | The Min property specifies the smallest possible value for the entropy computation. |
| **Max** | cyboxCommon:  FloatObjectPropertyType | 0..1 | The Max property specifies the largest possible value for the entropy computation (e.g., this would be 8 if the entropy computations is based on bits of information). |

## PEImportType Class

The PEImportType class is intended as container for the properties relevant to PE binary imports.

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

Table 3‑7. Properties of the PEImportType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **delay\_load** | basicDataTypes:Boolean | 0..1 | The delay\_load property is a boolean value that is intended to describe whether a PE binary import is delay-load or not. |
| **initially\_visible** | basicDataTypes:Boolean | 0..1 | The initially\_visible property refers to whether the import is initially visible, with regards to being initially visible or hidden in relation to PE binary packing. A packed binary will typically have few initially visible imports, and thus it is necessary to make the distinction between those that are visible initially or only after the binary is unpacked. |
| **File\_Name** | cyboxCommon:  StringObjectPropertyType | 0..1 | The File\_Name property specifies the name of the library (file) that the PE binary imports. |
| **Imported\_Functions** | PEImportedFunctionsType | 0..1 | The Imported\_Functions property is used to list any functions imported from a particular library. |
| **Virtual\_Address** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Virtual\_Address property specifies the relative virtual address (RVA) of the PE binary library import. |

## PEImportedFunctionsType Class

The PEImportedFunctionsType class captures a list of functions imported by the PE file.

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

Table 3‑8. Properties of the PEImportedFunctionsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Imported\_Function** | PEImportedFunctionType | 1..\* | Specifies a single property in a list of imported functions. |

## PEResourceType Class

The PEResourceType class is intended as container for the properties relevant to PE binary resources.

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

Table 3‑9. Properties of the PEResourceType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Type** | PEResourceContentType | 0..1 | The Type property refers to the type of data referred to by this resource. |
| **Name** | cyboxCommon:  StringObjectPropertyType | 0..1 | The Name property specifies the name of the resource used by the PE binary. |
| **Size** | cyboxCommon:  PositiveIntegerObjectPropertyType | 0..1 | The Size property specifies the size of the resource, in bytes. |
| **Virtual\_Address** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Virtual\_Address property specifies the relative virtual address (RVA) of the resource data. |
| **Language** | cyboxCommon:  StringObjectPropertyType | 0..1 | The Language property specifies the name of the language (LANG) defined for the resource, if applicable. |
| **Sub\_Language** | cyboxCommon:  StringObjectPropertyType | 0..1 | The Sub\_Language property specifies the name of the sub language (SUBLANG) defined for the resource, if applicable. |
| **Hashes** | cyboxCommon:HashListType | 0..1 | The Hashes property is used to include any hash values computed using the specified PE binary resource as input. |
| **Data** | cyboxCommon:  StringObjectPropertyType | 0..1 | The Data property captures the actual data contained in the resource, most commonly as a base64-encoded. |

## PEVersionInfoResourceType Class

The PEVersionInfoResourceType class characterizes the special VERSIONINFO resource class. For more information please see: <http://msdn.microsoft.com/en-us/library/windows/desktop/aa381058(v=vs.85).aspx>.

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

Table 3‑10. Properties of the PEVersionInfoResourceType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Comments** | cyboxCommon:  StringObjectPropertyType | 0..1 | The Comments property captures any additional information that should be displayed for diagnostic purposes. |
| **CompanyName** | cyboxCommon:  StringObjectPropertyType | 0..1 | The CompanyName property captures the company that produced the file - for example, "Microsoft Corporation" or "Standard Microsystems Corporation, Inc.". |
| **FileDescription** | cyboxCommon:  StringObjectPropertyType | 0..1 | The FileDescription property captures the file description to be presented to users. This string may be displayed in a list box when the user is choosing files to install - for example, "Keyboard Driver for AT-Style Keyboards". |
| **FileVersion** | cyboxCommon:  StringObjectPropertyType | 0..1 | The FileVersion property captures the version number of the file - for example, "3.10" or "5.00.RC2". |
| **InternalName** | cyboxCommon:  StringObjectPropertyType | 0..1 | The InternalName property captures the internal name of the file, if one exists - for example, a module name if the file is a dynamic-link library. If the file has no internal name, this string should be the original filename, without extension. |
| **LangID** | cyboxCommon:  StringObjectPropertyType | 0..1 | The LangID property captures the localization language identifier specified in the version-information resource. |
| **LegalCopyright** | cyboxCommon:  StringObjectPropertyType | 0..1 | The LegalCopyright property captures the copyright notices that apply to the file. This should include the full text of all notices, legal symbols, copyright dates, and so on. |
| **LegalTrademarks** | cyboxCommon:  StringObjectPropertyType | 0..1 | The LegalTrademarks property captures the trademarks and registered trademarks that apply to the file. This should include the full text of all notices, legal symbols, trademark numbers, and so on. |
| **OriginalFilename** | cyboxCommon:  StringObjectPropertyType | 0..1 | The OriginalFilename property captures the original name of the file, not including a path. This information enables an application to determine whether a file has been renamed by a user. The format of the name depends on the file system for which the file was created. |
| **PrivateBuild** | cyboxCommon:  StringObjectPropertyType | 0..1 | The PrivateBuild property captures the information about a private version of the file - for example, "Built by TESTER1 on \TESTBED". This string should be present only if VS\_FF\_PRIVATEBUILD is specified in the fileflags parameter of the root block. |
| **ProductName** | cyboxCommon:  StringObjectPropertyType | 0..1 | The ProductName property captures the name of the product with which the file is distributed. This property SHOULD be provided. |
| **ProductVersion** | cyboxCommon:  StringObjectPropertyType | 0..1 | The ProductVersion property captures the version of the product with which the file is distributed - for example, "3.10" or "5.00.RC2". |
| **SpecialBuild** | cyboxCommon:  StringObjectPropertyType | 0..1 | The SpecialBuild property captures the text that indicates how this version of the file differs from the standard version - for example, "Private build for TESTER1 solving mouse problems on M250 and M250E computers". This string should be present only if VS\_FF\_SPECIALBUILD is specified in the fileflags parameter of the root block. |

## PEExportedFunctionType Class

The PEExportedFunctionType class specifies the class describing exported functions.

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

Table 3‑11. Properties of the PEExportedFunctionType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Function\_Name** | cyboxCommon:  StringObjectPropertyType | 0..1 | The Function\_Name property specifies the name of the function exported by the PE binary. |
| **Entry\_Point** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Entry\_Point property specifies the entry point of the function exported by the PE binary. |
| **Ordinal** | cyboxCommon:  NonNegativeIntegerObjectPropertyType | 0..1 | The Ordinal property specifies the ordinal value (index) of the function exported by the PE binary. |

## PEResourceListType Class

The PEResourceListType class specifies a list of resources found in the PE file. The UML diagram corresponding to the PEResourceListType class is shown in Figure 3‑3**.**

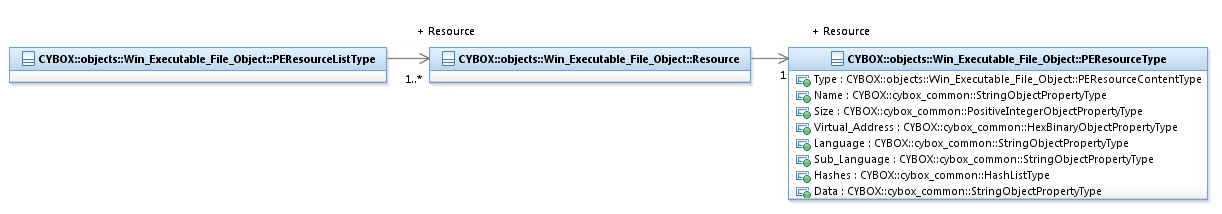


Figure 3‑3. UML diagram for the PEResourceListType class

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

Table 3‑12. Properties of the PEResourceListType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Resource** | PEResourceType | 1..\* | Specifies a property of a list of resources. |

## PEImportedFunctionType Class

The PEImportedFunctionType class specifies the class describing imported functions.

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

Table 3‑13. Properties of the PEImportedFunctionType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Function\_Name** | cyboxCommon:StringObjectPropertyType | 0..1 | The Function\_Name property specifies the name of the function from the specified library that the PE binary imports. |
| **Hint** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Hint property specifies the index into the export table of the library that the function is found in. |
| **Ordinal** | cyboxCommon:  NonNegativeIntegerObjectPropertyType | 0..1 | The Ordinal property specifies the ordinal value (index) of the function in the library that is found in. |
| **Bound** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Bound property specifies the precomputed address if the imported function is bound. |
| **Virtual\_Address** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Virtual\_Address property specifies the relative virtual address (RVA) of the PE binary library imported function. |

## PEImportListType Class

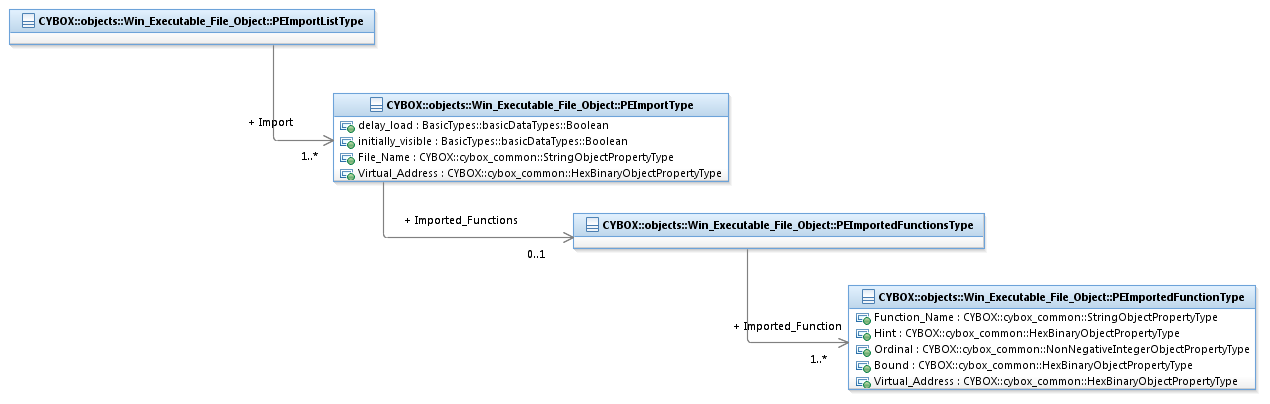
The PEImportListType class specifies a list of functions in an import data section. The UML diagram corresponding to the PEImportListType class is shown in Figure 3‑4**.**  


Figure 3‑4. UML diagram for the PEImportListType class

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

Table 3‑14. Properties of the PEImportListType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Import** | PEImportType | 1..\* | The Import property specifies a single property in a list of imported functions. |

## PESectionType Class

The PESectionType class is intended as container for the properties relevant to PE binary sections. A PE Section consists of a header and data. The PESectionType contains properties that describe the Section Header and metadata computed about the section (e.g., hashes, entropy). The UML diagram corresponding to the PESectionType class is shown in Figure 3‑5**.**

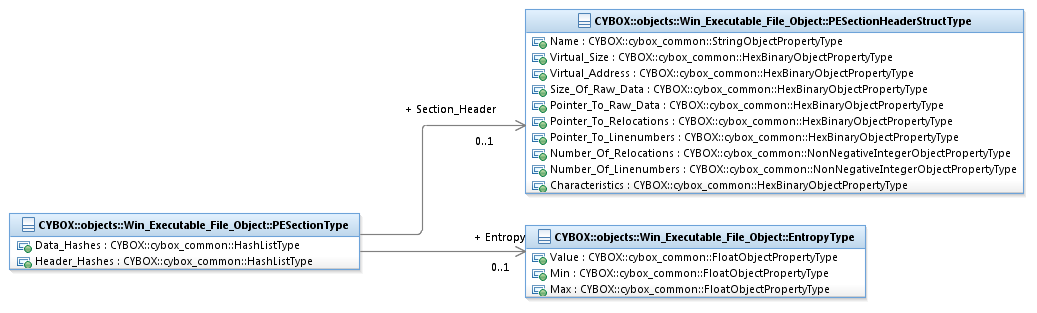


Figure 3‑5. UML diagram for the PESectionType class

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

Table 3‑15. Properties of the PESectionType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Section\_Header** | PESectionHeaderStructType | 0..1 | The Section\_Header property contains characteristics of the section's section header structure. |
| **Data\_Hashes** | cyboxCommon:HashListType | 0..1 | The Data\_Hashes property is used to include any hash values computed using the data contained in the specified PE binary section as input. |
| **Entropy** | EntropyType | 0..1 | The Entropy property specifies the calculated entropy of the PE binary section. |
| **Header\_Hashes** | cyboxCommon:HashListType | 0..1 | The Header\_Hashes property is used to include any hash values computed using the header of the specified PE binary section as input. |

## PEDataDirectoryStructType Class

The PEDataDirectoryStructType class is intended as container for the properties relevant to a PE binary's data directory structure.

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

Table 3‑16. Properties of the PEDataDirectoryStructType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Virtual\_Address** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Virtual\_Address property specifies the relative virtual address (RVA) of the data structure. |
| **Size** | cyboxCommon:  NonNegativeIntegerObjectPropertyType | 0..1 | The Size property specifies the size of the data structure, in bytes. |

## PESectionHeaderStructType Class

The PESectionHeaderStructType class is intended as container for the properties relevant to a PE binary's section header structure.

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

Table 3‑17. Properties of the PESectionHeaderStructType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Name** | cyboxCommon:  StringObjectPropertyType | 0..1 | The Name property specifies the name of the PE binary section. |
| **Virtual\_Size** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Virtual\_Size property is the total size of the PE binary section when loaded into memory. It is valid only for executables and should be 0 for object files. |
| **Virtual\_Address** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Virtual\_Address property specifies the relative virtual address (RVA) of the PE binary section. |
| **Size\_Of\_Raw\_Data** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Size\_Of\_Raw\_Data property specifies the size of the data contained in the PE binary section. |
| **Pointer\_To\_Raw\_Data** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Pointer\_To\_Raw\_Data property specifies the file offset of the beginning of the PE binary section. |
| **Pointer\_To\_Relocations** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Pointer\_To\_Relocations property specifies the offset of the PE binary section relocations, if applicable. |
| **Pointer\_To\_Linenumbers** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Pointer\_To\_Linenumbers property specifies the beginning of line-number entries for the section. Should be 0. |
| **Number\_Of\_Relocations** | cyboxCommon:  NonNegativeIntegerObjectPropertyType | 0..1 | The Number\_Of\_Relocations property specifies the number of relocations defined for the specified PE binary section. |
| **Number\_Of\_Linenumbers** | cyboxCommon:  NonNegativeIntegerObjectPropertyType | 0..1 | The Number\_Of\_Linenumbers property specifies the number of line number entries for the section. Should be 0. |
| **Characteristics** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Characteristics property specifies any flags defined for the specified PE binary section. |

## DOSHeaderType Class

The DOSHeaderType class is a container for the characteristics of the \_IMAGE\_DOS\_HEADER structure, which can be found in Winnt.h and pe.h. See <http://www.csn.ul.ie/~caolan/pub/winresdump/winresdump/doc/pefile.html> for more information about the winnt.h file, and [http://www.tavi.co.uk/phobos/exeformat.html](http://www.tavi.co.uk/phobos/exeformat.html%20) for even more clarification.

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

Table 3‑18. Properties of the DOSHeaderType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **e\_magic** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_magic property specifies the magic number, specifically the Windows OS signature value, which can either take on MZ for DOS (which is, for all intents and purposes, most Windows executables), NE for OS2, LE for OS2 LE, or PE00 for NT. |
| **e\_cblp** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_cblp property specifies the number of bytes actually used in the last page, with the special case of a full page being represented by a value of zero (since the last page is never empty). For example, assuming a page size of 512 bytes, this value would be 0x0000 for a 1024 byte file, and 0x0001 for a 1025 byte file (since it only contains one valid byte). |
| **e\_cp** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_cp property specifies the number of pages required to hold the file. For example, if the file contains 1024 bytes, and we assume the file has pages of a size of 512 bytes, this word would contain 0x0002; if the file contains 1025 bytes, this word would contain 0x0003. |
| **e\_crlc** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_crlc property specifies the number of relocation items, i.e. the number of entries that exist in the relocation pointer table. If there are no relocation entries, this value is zero. |
| **e\_cparhdr** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_cparhdr property specifies the size of the executable header in terms of paragraphs (16 byte chunks). It indicates the offset of the program's compiled/assembled and linked image (the load module) within the executable file. The size of the load module can be deduced by subtracting this value (converted to bytes) from the overall file size derived from combining the e\_cp (number of file pages) and e\_cblp (number of bytes in last page) values. The header always spans an even number of paragraphs. |
| **e\_minalloc** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_minalloc property specifies the minimum number of extra paragraphs needed to be allocated to begin execution. This is *in addition* to the memory required to hold the load module. This value normally represents the total size of any uninitialized data and/or stack segments that are linked at the end of a program. This space is not directly included in the load module, since there are no particular initializing values and it would simply waste disk space. |
| **e\_maxalloc** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_maxalloc property specifies the maximum number of extra paragraphs needed to be allocated by the program before it begins execution. This indicates *additional* memory over and above that required by the load module and the value specified by MINALLOC. If the request cannot be satisfied, the program is allocated as much memory as is available. |
| **e\_ss** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_ss property specifies the initial SS value, which is the paragraph address of the stack segment relative to the start of the load module. At load time, this value is relocated by adding the address of the start segment of the program to it, and the resulting value is placed in the SS register before the program is started. In DOS, the start segment of the program is the first segment boundary in memory after the PSP. |
| **e\_sp** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_sp property specifies the initial SP value, which is the absolute value that must be loaded into the SP register before the program is given control. Since the actual stack segment is determined by the loader, and this is merely a value within that segment, it does not need to be relocated. |
| **e\_csum** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_csum property specifies the checksum of the contents of the executable file. It is used to ensure the integrity of the data within the file. For full details on how this checksum is calculated, see [http://www.tavi.co.uk/phobos/exeformat.html#checksum](http://www.tavi.co.uk/phobos/exeformat.html%23checksum). |
| **e\_ip** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_ip property specifies the initial IP value, which is the absolute value that should be loaded into the IP register in order to transfer control to the program. Since the actual code segment is determined by the loader, and this is merely a value within that segment, it does not need to be relocated. |
| **e\_cs** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_cs property specifies the pre-relocated initial CS value, relative to the start of the load module that should be placed in the CS register in order to transfer control to the program. At load time, this value is relocated by adding the address of the start segment of the program to it, and the resulting value is placed in the CS register when control is transferred. |
| **e\_lfarlc** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_lfarlc property specifies the file address of the relocation table, or more specifically, the offset from the start of the file to the relocation pointer table. This value must be used to locate the relocation pointer table (rather than assuming a fixed location) because variable-length information pertaining to program overlays can occur before this table, causing its position to vary. A value of 0x40 in this property generally indicates a different kind of executable file, not a DOS 'MZ' type. |
| **e\_ovro** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_ovro property specifies the overlay number, which is normally set to 0x0000, because few programs actually have overlays. It changes only in files containing programs that use overlays. See [http://www.tavi.co.uk/phobos/exeformat.html#overlaynote](http://www.tavi.co.uk/phobos/exeformat.html%23overlaynote%20) for more information about overlays. |
| **reserved1** | cyboxCommon:  HexBinaryObjectPropertyType | 0..4 | The reserved1 property specifies reserved words for the program (known in winnt.h as e\_res[4]), usually set to zero by the linker. In this case, just use a single reserved1 set to zero; if not zero create four reserved1 with the correct value. |
| **e\_oemid** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_oemid property specifies the identifier for the OEM for e\_oeminfo. |
| **e\_oeminfo** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_oeminfo property specifies the OEM information for a specific value of e\_oeminfo. |
| **reserved2** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The reserved2 property specifies reserved words for the program (known in winnt.h as e\_res[10]), usually set to zero by the linker. In this case, just use a single reserved1 set to zero; if not zero create ten reserved1 with the correct value. |
| **e\_lfanew** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The e\_lfanew property specifies the file address of the of the new exe header. In particular, it is a 4-byte offset into the file where the PE file header is located. It is necessary to use this offset to locate the PE header in the file. |
| **Hashes** | cyboxCommon:HashListType | 0..1 | The Hashes property is used to include any hash values computed using the specified PE binary MS-DOS header as input. |

## PEHeadersType Class

The PEHeadersType class specifies the headers found in PE and COFF files. The UML diagram corresponding to the PEHeadersType class is shown in Figure 3‑6**.**

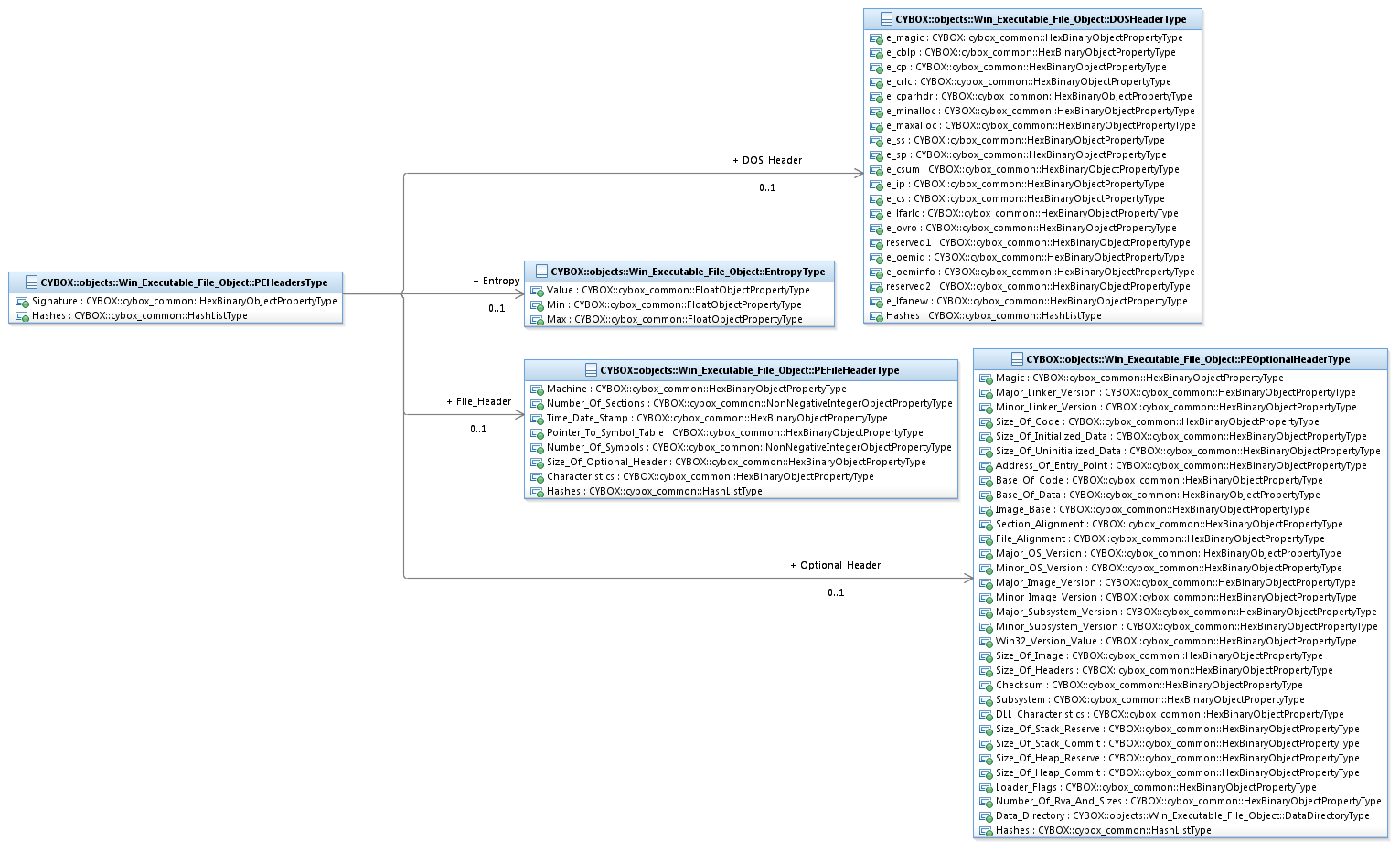


Figure 3‑6. UML diagram for the PEHeaderType class

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

Table 3‑19. Properties of the PEHeadersType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **DOS\_Header** | DOSHeaderType | 0..1 | The DOS\_Header property refers to the MS-DOS PE header and its associated characteristics. |
| **Signature** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Signature property specifies the 4-bytes signature that identifies the file as a PE file. |
| **File\_Header** | PEFileHeaderType | 0..1 | The File\_Header property refers to the PE file header (sometimes referred to as the COFF header) and its associated characteristics. |
| **Optional\_Header** | PEOptionalHeaderType | 0..1 | The Optional\_Header property refers to the PE optional header and its associated characteristics. The Optional Header is required for executable (PE) files, but optional for object (COFF) files. |
| **Entropy** | EntropyType | 0..1 | The Entropy property specifies the calculated entropy of the PE file header. |
| **Hashes** | cyboxCommon:HashListType | 0..1 | The Hashes property is used to include any hash values computed using the specified PE binary file header as input. |

## PEFileHeaderType Class

The PEFileHeaderType class refers to the PE file header (sometimes referred to as the COFF header) and its associated characteristics.

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

Table 3‑20. Properties of the PEFileHeaderType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Machine** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Machine property specifies the type of target machine. |
| **Number\_Of\_Sections** | cyboxCommon:  NonNegativeIntegerObjectPropertyType | 0..1 | The Number\_Of\_Sections property specifies the number of sections in the file. |
| **Time\_Date\_Stamp** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Time\_Date\_Stamp property specifies the time when the file was created (the low 32 bits of the number of seconds since epoch). |
| **Pointer\_To\_Symbol\_Table** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Pointer\_To\_Symbol\_Table property specifies the file offset of the COFF symbol table (should be 0). |
| **Number\_Of\_Symbols** | cyboxCommon:  NonNegativeIntegerObjectPropertyType | 0..1 | The Number\_Of\_Symbols property specifies the number of entries in the symbol table (should be 0). |
| **Size\_Of\_Optional\_Header** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Size\_Of\_Optional\_Header property specifies the size of the optional header. It should be 0 for object files and non-zero for executables. |
| **Characteristics** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Characteristics property specifies the flags that indicate the file's characteristics. |
| **Hashes** | cyboxCommon:HashListType | 0..1 | The Hashes property specifies any hashes computed for the Optional Header. |

## SubsystemType Data Type

The SubsystemType data type specifies the subsystem type. Its core value SHOULD be a literal from the SubsystemTypeEnum enumeration. It extends the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

## PEType Data Type

The PEType data type specifies the PE file type. Its core value SHOULD be a literal from the PETypeEnum enumeration. It extends the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

## PEOptionalHeaderType Class

The PEOptionalHeaderType class describes the PE Optional Header structure. Additional computed metadata, e.g., hashes of the header, are also included.

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

Table 3‑21. Properties of the PEOptionalHeaderType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Magic** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Magic property specifies the unsigned integer that indicates the type of executable file. |
| **Major\_Linker\_Version** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Major\_Linker\_Version property specifies the linker major version number. |
| **Minor\_Linker\_Version** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Minor\_Linker\_Version property specifies the linker minor version number. |
| **Size\_Of\_Code** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Size\_Of\_Code property specifies the size of the code (text) section. If there are multiple sections, size is the sum of the sizes if each. |
| **Size\_Of\_Initialized\_Data** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Size\_Of\_Initialized\_Data property specifies the size of the initialized data section. If there are multiple sections, size is the sum of the sizes if each. |
| **Size\_Of\_Uninitialized\_Data** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Size\_Of\_Uninitialized\_Data property specifies the size of the uninitialized (bss) data section. If there are multiple sections, size is the sum of the sizes if each. |
| **Address\_Of\_Entry\_Point** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Address\_Of\_Entry\_Point property specifies the address of the entry point relative to the image base when the executable is loaded into memory. When there is no entry point (e.g., optional for DLLs), the value should be 0. |
| **Base\_Of\_Code** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Base\_Of\_Code property specifies the address that is relative to the image base of the beginning-of-code section when it is loaded into memory. |
| **Base\_Of\_Data** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Base\_Of\_Data property specifies the address that is relative to the image base of the beginning-of-data section when it is loaded into memory. |
| **Image\_Base** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Image\_Base property specifies the preferred address of the first byte of image when loaded into memory; must be a multiple of 64 K. |
| **Section\_Alignment** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Section\_Alignment property specifies the alignment (in bytes) of sections when they are loaded into memory. |
| **File\_Alignment** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The File\_Alignment property specifies the factor (in bytes) that is used to align the raw data of sections in the image file. |
| **Major\_OS\_Version** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Major\_OS\_Version property specifies the major version number of the required operating system. |
| **Minor\_OS\_Version** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Minor\_OS\_Version property specifies the minor version number of the required operating system. |
| **Major\_Image\_Version** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Major\_Image\_Version property specifies the major version number of the image. |
| **Minor\_Image\_Version** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Minor\_Image\_Version property specifies the minor version number of the image. |
| **Major\_Subsystem\_Version** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Major\_Subsystem\_Version property specifies the major version number of the subsystem. |
| **Minor\_Subsystem\_Version** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Minor\_Subsystem\_Version property specifies the minor version number of the subsystem. |
| **Win32\_Version\_Value** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Win32\_Version\_Value property is reserved; must be 0. |
| **Size\_Of\_Image** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Size\_Of\_Image property specifies the size (in bytes) of the image, including all headers, as the image is loaded in memory. |
| **Size\_Of\_Headers** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Size\_Of\_Headers property specifies the combined size of the MS DOS header, PE header, and section headers rounded up to a multiple of FileAlignment. |
| **Checksum** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Checksum property specifies the checksum of the PE file. |
| **Subsystem** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Subsystem property specifies the subsystem (e.g., GUI, device driver) that is required to run this image. |
| **DLL\_Characteristics** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The DLL\_Characteristics property specifies flags that characterize the PE file. |
| **Size\_Of\_Stack\_Reserve** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Size\_Of\_Stack\_Reserve property specifies the size of the stack to reserve. |
| **Size\_Of\_Stack\_Commit** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Size\_Of\_Stack\_Commit property specifies the size of the stack to commit. |
| **Size\_Of\_Heap\_Reserve** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Size\_Of\_Heap\_Reserve property specifies the size of the local heap space to reserve. |
| **Size\_Of\_Heap\_Commit** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Size\_Of\_Heap\_Commit property specifies the size of the local heap space to commit. |
| **Loader\_Flags** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Loader\_Flags property is reserved; must be 0. |
| **Number\_Of\_Rva\_And\_Sizes** | cyboxCommon:  HexBinaryObjectPropertyType | 0..1 | The Number\_Of\_Rva\_And\_Sizes property specifies the number of data-directory entries in the remainder of the optional header. |
| **Data\_Directory** | DataDirectoryType | 0..1 | The Data\_Directory property specifies the data directories in the remainder in the optional header. |
| **Hashes** | cyboxCommon:  HashListType | 0..1 | The Hashes property is used to include any hash values computed using the specified PE binary optional header as input. |

## DataDirectoryType Class

The DataDirectoryType class specifies the data directories that can appear in the PE file's optional header. The data directories, except the Certificate Table, are loaded into memory so they can be used at runtime.

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

Table 3‑22. Properties of the DataDirectoryType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Export\_Table** | PEDataDirectoryStructType | 0..1 | The Export\_Table property specifies the export table data directory. |
| **Import\_Table** | PEDataDirectoryStructType | 0..1 | The Import\_Table property specifies the import table data directory. |
| **Resource\_Table** | PEDataDirectoryStructType | 0..1 | The Resource\_Table property specifies the resource table data directory. |
| **Exception\_Table** | PEDataDirectoryStructType | 0..1 | The Exception\_Table property specifies the exception table data directory. |
| **Certificate\_Table** | PEDataDirectoryStructType | 0..1 | The Certificate\_Table property specifies the certificate table data directory. The table of certificates is in a file which the data directory points to. |
| **Base\_Relocation\_Table** | PEDataDirectoryStructType | 0..1 | The Base\_Relocation\_Table property specifies the base relocation table data directory. |
| **Debug** | PEDataDirectoryStructType | 0..1 | The Debug property specifies the debug data directory. |
| **Architecture** | PEDataDirectoryStructType | 0..1 | The Architecture property is reserved, must be 0. |
| **Global\_Ptr** | PEDataDirectoryStructType | 0..1 | The Global\_Ptr property specifies the RVA of the value to be stored in the global pointer register. |
| **TLS\_Table** | PEDataDirectoryStructType | 0..1 | The TLS\_Table property specifies the thread local storage (TLS) table data directory. |
| **Load\_Config\_Table** | PEDataDirectoryStructType | 0..1 | The Load\_Config\_Table property specifies the load configuration table data directory. |
| **Bound\_Import** | PEDataDirectoryStructType | 0..1 | The Bound\_Import property specifies the bound import table data directory. |
| **Import\_Address\_Table** | PEDataDirectoryStructType | 0..1 | The Import\_Address\_Table property specifies the import address table (IAT) data directory. |
| **Delay\_Import\_Descriptor** | PEDataDirectoryStructType | 0..1 | The Delay\_Import\_Descriptor property specifies the delay import descriptor data directory. |
| **CLR\_Runtime\_Header** | PEDataDirectoryStructType | 0..1 | The CLR\_Runtime\_Header property specifies the Common Language Runtime (CLR) header data directory. |
| **Reserved** | PEDataDirectoryStructType | 0..1 | The Reserved property is reserved; must be 0. |

## PEBuildInformationType Class

The PEBuildInformationType class captures information about the tools used to build the PE binary, including the compiler and linker.

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

Table 3‑23. Properties of the PEBuildInformationType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Linker\_Name** | cyboxCommon:  StringObjectPropertyType | 0..1 | The Linker\_Name property specifies the name of the linker used to link the PE binary. |
| **Linker\_Version** | cyboxCommon:  StringObjectPropertyType | 0..1 | The Linker\_Version property specifies the version of the linker used to link the PE binary. |
| **Compiler\_Name** | cyboxCommon:  StringObjectPropertyType | 0..1 | The Compiler\_Name property specifies the name of the compiler used to compile the binary. |
| **Compiler\_Version** | cyboxCommon:  StringObjectPropertyType | 0..1 | The Compiler\_Version property specifies the version of the compiler used to compile the binary. |

## PEResourceContentType Data Type

The PEResourceContentType data type specifies the PE resource type. Its core value SHOULD be a literal from the PEResourceTypeEnum enumeration. It extends the BaseObjectPropertyType data type, in order to permit complex (i.e. regular-expression based) specifications.

## SubsystemTypeEnum Enumeration

The literals of the SubsystemTypeEnum enumeration are given in **Table 3‑24**.

Table 3‑24. Literals of the SubsystemTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **Unknown** | Specifies an unknown subsystem. |
| **Native** | Specifies that no subsystem is required to run the image (i.e. only device drivers and native system processes are needed). |
| **Windows\_GUI** | Specifies the Windows Graphical user interface (GUI) subsystem. |
| **Windows\_CUI** | Specifies the Windows character-mode user interface (CUI) subsystem. |
| **OS2\_CUI** | Specifies the OS/2 CUI subsystem. |
| **POSIX\_CUI** | Specifies the POSIX CUI subsystem. |
| **Native\_Win9x\_Driver** | Specifies the Native Windows 9x drivers. This is denoted by the value IMAGE\_SUBSYSTEM\_NATIVE\_WINDOWS or 0x8. |
| **Windows\_CE\_GUI** | Specifies the Windows CE system with a GUI. |
| **EFI\_Application** | Specifies the Extensible Firmware Interface (EFI) application. |
| **EFI\_Boot\_Service\_Driver** | Specifies the Extensible Firmware Interface (EFI) driver with boot services. |
| **EFI\_Runtime\_Driver** | Specifies the Extensible Firmware Interface (EFI) driver with run-time services. |
| **EFI\_ROM** | Specifies the Extensible Firmware Interface (EFI) image. |
| **XBOX** | Specifies the XBOX system. |
| **Windows\_Boot\_Application** | Specifies the Windows Boot application. |

## PETypeEnum Enumeration

The literals of the PETypeEnum enumeration are given in **Table 3‑25**.

Table 3‑25. Literals of the PETypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **Executable** | Specifies an executable image (not an OBJ or LIB). |
| **Dll** | Specifies a dynamic link library, not a program. |
| **Invalid** | Specifies an invalid executable file (i.e. not one of the listed types). |

## PEResourceTypeEnum Enumeration

The literals of the PEResourceTypeEnum enumeration are given in **Table 3‑26**.

Table 3‑26. Literals of the PEResourceTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **Cursor** | Specifies a resource that is a cursor or animated cursor defined by naming it and specifying the name of the file that contains it. (To use a particular cursor, the application requests it by name.). |
| **Bitmap** | Specifies a resource that is a bitmap defined by naming it and specifying the name of the file that contains it. (To use a particular cursor, the application requests it by name.). |
| **Icon** | Specifies a resource that is an icon or animated icon by naming it and specifying the name of the file that contains it. (To use a particular icon, the application requests it by name.). |
| **Menu** | Specifies a resource that captures the appearance and function of a menu. Does not define help or regular identifiers, nor uses the MFT\_\* type and MFS\_\* state flags. |
| **MenuEX** | Specifies a resource that captures the appearance and function of a menu, which can also utilize help or regular identifiers, as well as the MFT\_\* type and MFS\_\* state flags. |
| **Popup** | Specifies a resource that captures a menu item that can contain menu items and submenus. |
| **Dialog** | Specifies a resource that captures a template that an application can use to create dialog boxes. This type is considered obsolete in Windows and newer applications use the DIALOGEX resource. |
| **DialogEX** | Specifies a resource that captures a template that newer applications can use to create dialog boxes. |
| **String** | Specifies a resource that is a string. |
| **StringTable** | Specifies a resource that captures string tables. String resources are Unicode or ASCII strings that can be loaded from the executable file. |
| **Fontdir** | Specifies a resource that is a font directory. |
| **Font** | Specifies a resource that captures the name of a file that contains a font. |
| **Accelerators** | Specifies a resource that captures menu accelerator keys. |
| **RCData** | Specifies a resource that captures data resources. Data resources let you include binary data in the executable file. |
| **MessageTable** | Specifies a resource that captures a message table by naming it and specifying the name of the file that contains it. The file is a binary resource file generated by the message compiler. |
| **GroupCursor** | Specifies a resource that is a group cursor. |
| **GroupIcon** | Specifies a resource that is a group icon. |
| **VersionInfo** | Specifies a resource that captures version-information. Contains information such as the version number, intended operating system, and so on. |
| **DLGInclude** | Specifies a resource that is a dialog include. |
| **PlugPlay** | This resource is obsolete and included for completeness. |
| **TextInclude** | This is a special resource that is interpreted by Visual C++. For more information see http://go.microsoft.com/FWLink/?LinkId=83951. |
| **TypeLib** | This is a special resource that is used with /TLBID and /TLBOUT linker options. For more information see <http://go.microsoft.com/FWLink/?LinkId=83960> (for /TLBID) and [http://go.microsoft.com/FWLink/?LinkId=83947](http://go.microsoft.com/FWLink/?LinkId=83947%20) (for /TLBOUT). |
| **Vxd** | This resource is obsolete and included for completeness. |
| **AniCursor** | Specifies a resource that is an animated cursor. |
| **AniIcon** | Specifies a resource that is an animated icon. |
| **HTML** | Specifies a resource that captures an HTML file. |
| **Manifest** | Specifies a resource that captures a manifest file. |
| **MessageTableEntry** | Specifies a resource that captures a message table entry. |

# 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.

Acknowledgments

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

Participants:

Dean Thompson, Australia and New Zealand Banking Group (ANZ Bank)

Bret Jordan, Blue Coat Systems, Inc.

Adnan Baykal, Center for Internet Security (CIS)

Liron Schiff, Comilion (mobile) Ltd.

Jane Ginn, Cyber Threat Intelligence Network, Inc. (CTIN)

Richard Struse, DHS Office of Cybersecurity and Communications (CS&C)

Ryusuke Masuoka, Fujitsu Limited

Eric Burger, Georgetown University

Jason Keirstead, IBM

Paul Martini, iboss, Inc.

Jerome Athias, Individual

Sanjiv Kalkar, Individual

Terry MacDonald, Individual

Alex Pinto, Individual

Patrick Maroney, Integrated Networking Technologies, Inc.

Wouter Bolsterlee, Intelworks BV

Joep Gommers, Intelworks BV

Sergey Polzunov, Intelworks BV

Rutger Prins, Intelworks BV

Andrei Sîrghi, Intelworks BV

Jonathan Baker, MITRE Corporation

Sean Barnum, MITRE Corporation

Mark Davidson, MITRE Corporation

Ivan Kirillov, MITRE Corporation

John Wunder, MITRE Corporation

Mike Boyle, National Security Agency

Jessica Fitzgerald-McKay, National Security Agency

Takahiro Kakumaru, NEC Corporation

John-Mark Gurney, New Context Services, Inc.

Christian Hunt, New Context Services, Inc.

Andrew Storms, New Context Services, Inc.

Igor Baikalov, Securonix

Bernd Grobauer, Siemens AG

John Anderson, Soltra

Trey Darley, Soltra

Paul Dion, Soltra

Brandon Hanes, Soltra

Ali Khan, Soltra

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

Revision History

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