CybOX Version 2.1.1 Part 3: Core

Working Draft 01

28 August 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:

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

Desiree Beck ([dbeck@mitre.org](mailto:dbeck@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 2.2.1 Part 1: Overview*. [URI – added during publication]
* *CybOX Version 2.2.1 Part 2: Common*. [URI]
* *CybOX Version 2.2.1 Part 3: Core*. [URI]
* *CybOX Version 2.2.1 Part XXX: UML Model*. [URI]

Related work:

This specification is related to:

* *STIX 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 Core data model, which is one of the fundamental 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/part9-coa/stix-v1.2.1-csd01-part9-coa.docx

Permanent “Latest version” URI:  
http://docs.oasis-open.org/cti/stix/v1.2.1/stix-v1.2.1-part9-coa.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.

Table of Contents

[1 Introduction 4](#_Toc409437255)

[1.1 Terminology 4](#_Toc409437256)

[1.2 Normative References 4](#_Toc409437257)

[1.3 Non-Normative References 4](#_Toc409437258)

[2 Section Title 5](#_Toc409437259)

[2.1 Level 2 section title 5](#_Toc409437260)

[2.1.1 Level 3 section title 5](#_Toc409437261)

[2.1.1.1 Level 4 section title is usually deepest for Table of Contents 5](#_Toc409437262)

[3 # Conformance 6](#_Toc409437263)

[Appendix A. Acknowledgments 7](#_Toc409437264)

[Appendix B. Example Title 8](#_Toc409437265)

[B.1 Subsidiary section 8](#_Toc409437266)

[B.1.1 Sub-subsidiary section 8](#_Toc409437267)

[B.1.1.1 Sub-sub-subsidiary section 8](#_Toc409437268)

[Appendix C. Revision History 9](#_Toc409437269)

# Introduction

[All text is normative unless otherwise labeled]

The Cyber Observable Expression (CybOX) 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 Core Version 2.1.1 data model, which is one of two fundamental data models for CybOX content.

In Section **1.1** we discuss additional specification documents, in Section **1.2** we provide document conventions in, and in Section **1.3** we provide terminology. References are given in Sections **1.4** and **1.5**. In Section **2**, we give background information necessary to fully understand the Course of Action data model. We present the Core data model specification details in Section **3** and conformance information in Section **4**.

## CybOX 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 key 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](#AdditionalArtifacts) is not necessary.

The [*CybOX Version 2.2.1 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 a set of default controlled vocabularies. [*CybOX Version 2.2.1 Part 1: Overview*](#AdditionalArtifacts) also summarizes the relationship of CybOX to other externally defined data models, 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 2.2.1 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](#AdditionalArtifacts). 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. [*CybOX Version 2.2.1 Part 1: Overview*](#AdditionalArtifacts)contains the full list of CybOX packages, along with the associated prefix notations, descriptions, and examples.

Note that in *this* specification document, we do not explicitly specify the package prefix for any classes that originate from the Core 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. For example, properties of a class that are identifiers, titles, and timestamps will be represented 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 ‑. UML diagram icons

|  |  |
| --- | --- |
| **Icon** | **Description** |
| cid:image003.gif@01D05428.2B30AE20 | 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. |

#### Color Coding

The shapes of the UML diagrams are color coded to indicate the data model associated with a class.  The colors used in the Core specification are illustrated via exemplars in **Figure 1‑1**.

[need diagram]

Figure ‑. Data model color coding

### 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 Core 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.

In addition, properties that are part of a “choice” relationship (e.g., Prop1 OR Prop2 is used but not both) will be denoted by a unique letter subscript (e.g., API\_CallA, CodeB) and single logic expression in the Multiplicity column.  For example, if there is a choice of property API\_CallA and CodeB, the expression “A(1)|B(0..1)” will indicate that the API\_Call property can be chosen with multiplicity 1 or the Code property can be chosen with multiplicity 0 or 1.

### 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 chosen 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 identitifying 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>.

## Non-Normative References

# Background Information

In this section, we provide high level information about the Core 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.

# 

# CybOX Core Data Model

The CybOX Core data model defines a variety of classes. For discussion purposes, we have separated the classes into six categories (Sections **3.1** through **3.5**), and within each category, we primarily define the classes in alphabetical order below, except for the cases when one class (a superclass) is specialized by other classes, in which case the superclass is defined first (and the other classes are either listed alphabetically or in another order as explained). We list enumerations in Section **3.6**.

## Primary Classes

The following classes are the primary classes in CybOX and enable the capture of Actions, Events, Objects, and Observables (Stateful Measures).

### ActionType Class

The ActionType class characterizes a cyber observable Action.

Table ‑. Properties of the ActionType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **id** | basicDataTypes:  QualifiedName | 0..1 | The id property specifies a globally unique identifier for the Action. |
| **idref** | basicDataTypes:  QualifiedName | 0..1 | The idref property specifies a globally unique identifier for an Action specified elsewhere. When the idref property is used, the id property MUST NOT also be specified and the other properties of the ActionType class SHOULD NOT hold any content. |
| **ordinal\_position** | basicDataTypes:  PositiveInteger | 0..1 | The ordinal\_position property specifies the order (e.g., ‘*1’*, ‘*2’*, or ‘*3’*) of the Action within a potential set of multiple Actions. If only a single Action is present, its ordinality can be assumed to be 1. If multiple Actions are present, the ordinality property SHOULD be specified with unique values for each instance. |
| **action\_status** | ActionStatusTypeEnum | 0..1 | The action\_status property specifies the status of the Action. |
| **context** | ActionContextTypeEnum | 0..1 | The context property specifies the broad operational context in which the Action is relevant. |
| **timestamp** | basicDataTypes:  DateTime | 0..1 | The timestamp property specifies the date and time at which the Action occurred or was observed. To avoid ambiguity, all timestamps SHOULD include a specification of the time zone. |
| **timestamp\_precision** | cyboxCommon:  DateTimePrecisionEnum | 0..1 | The timestamp\_precision property specifies the granularity with which the timestamp property should be considered, as specified by the DateTypePrecisionEnum enumeration (e.g., '*hour*,' '*minute*'). If omitted, the default precision is ‘*second*.’ Digits in a timestamp that are beyond the specified precision should be zeroed out. |
| **Type** | cyboxCommon:  VocabularyStringType | 0..1 | The Type property specifies the type of the Action that was performed. Examples of potential types include *compress, replicate*, and *suspend* (these specific values are only provided to help explain the property: they are neither recommended values nor necessarily part of any existing vocabulary). The content creator may choose any arbitrary value or may constrain the set of possible values by referencing an externally-defined vocabulary or leveraging a formally defined vocabulary extending from the cyboxCommon:ControlledVocabularyStringType class. The CybOX default vocabulary class for use in the property is ‘*ActionTypeVocab-1.0*’. |
| **Name** | cyboxCommon:  VocabularyStringType | 0..1 | The Name property specifies the name of the Action that was performed. Examples of potential names include *add user, connect to socket*, and *monitor resgistry key* (these specific values are only provided to help explain the property: they are neither recommended values nor necessarily part of any existing vocabulary). The content creator may choose any arbitrary value or may constrain the set of possible values by referencing an externally-defined vocabulary or leveraging a formally defined vocabulary extending from the cyboxCommon:ControlledVocabularyStringType class. The CybOX default vocabulary class for use in the property is ‘*ActionNameVocab-1.1*’. |
| **Description** | cyboxCommon:  StructuredTextType | 0..1 | The Description property captures a textual description of the Action. Any length is permitted. Optional formatting is supported via the structuring\_format property of the StructuredTextType class. |
| **Action\_Aliases** | ActionAliasesType | 0..1 | The Action\_Aliases property specifies a set of one or more alias names for the Action. |
| **Action\_Arguments** | ActionArgumentsType | 0..1 | The Action\_Arguments property specifies a set of one or more arguments or parameters relevant to the Action. |
| **Location** | cyboxCommon:  LocationType | 0..1 | The Location property characterizes the actual physical location of the Object. A simple location name may be specified or the underlying abstract class may be extended, in which case, the default and strongly RECOMMENDED subclass is CIQAddress3.0InstanceType, as defined in *[CybOX Version 2.2.1 Part XXX: Default Extensions](#AdditionalArtifacts)*. |
| **Discovery\_Method** | cyboxCommon:  MeasureSourceType | 0..1 | The Discovery\_Method property characterizes how the Action was observed (in the case of a cyber observable Action instance) or could potentially be observed (in the case of a cyber observable Action pattern). Examples of details captured include identifying characteristics, time-related attributes, and a list of the tools used to collect the information. |
| **Associated\_Objects** | AssociatedObjectsType | 0..1 | The Associated\_Objects property specifies a set of one or more cyber Objects relevant to the Action (initiating or affected by). |
| **Relationships** | ActionRelationshipsType | 0..1 | The Relationships property specifies a set of one or more relationships between this Action and other Actions. |
| **Frequency** | FrequencyType | 0..1 | The Frequency property characterizes the frequency of the Action. |

### EventType Class

The EventType class characterizes a cyber observable Event, which is a set of specific Action(s) involving specific cyber relevant Objects. Examples of Events include: a file is deleted, a registry key is created, or an HTTP GET Request is received.

Table ‑. Properties of the EventType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **id** | basicDataTypes:  QualifiedName | 0..1 | The id property specifies a globally unique identifier for the Event. |
| **idref** | basicDataTypes:  QualifiedName | 0..1 | The idref property specifies a globally unique identifier for an Event specified elsewhere. When the idref property is used, the id property MUST NOT also be specified and the other properties of the EventType class SHOULD NOT hold any content. |
| **Type** | cyboxCommon:  VocabularyStringType | 0..1 | The Type property specifies the type of the Event. Examples of potential types include *http traffic, socket operations*, and *autorun* (these specific values are only provided to help explain the property: they are neither recommended values nor necessarily part of any existing vocabulary). The content creator may choose any arbitrary value or may constrain the set of possible values by referencing an externally-defined vocabulary or leveraging a formally defined vocabulary extending from the cyboxCommon:ControlledVocabularyStringType class. The CybOX default vocabulary class for use in the property is ‘*EventTypeVocab-1.0.1’*. |
| **Description** | cyboxCommon:  StructuredTextType | 0..1 | The Description property captures a textual description of the Event. Any length is permitted. Optional formatting is supported via the structuring\_format property of the StructuredTextType class. |
| **Observation\_Method** | cyboxCommon:  MeasureSourceType | 0..1 | The Observation\_Method property characterizes how the Event was observed (in the case of a cyber observable Event instance) or could potentially be observed (in the case of a cyber observable Event pattern). Examples of details captured include identifying characteristics, time-related attributes, and a list of the tools used to collect the information |
| **Actions** | ActionsType | 0..1 | The Actions property specifies a set of one or more Actions that compose the Event. |
| **Location** | cyboxCommon:  LocationType | 0..1 | The Location property characterizes the actual physical location of the Object. A simple location name may be specified or the underlying abstract class may be extended, in which case, the default and strongly RECOMMENDED subclass is CIQAddress3.0InstanceType, as defined in *[CybOX Version 2.2.1 Part XXX: Default Extensions](#AdditionalArtifacts)*. |
| **Frequency** | FrequencyType | 0..1 | The Frequency property characterizes the frequency of the Event. |
| **Event** | EventType | 1..\* | The Event property specifies a set of one or more Events that define a composite Event. |

### ObjectType Class

The ObjectType class characterizes a cyber-relevant Object (e.g., a file, a registry key or a process).

Table ‑. Properties of the ObjectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **id** | basicDataTypes:  QualifiedName | 0..1 | The id property specifies a globally unique identifier for the Object. |
| **idref** | basicDataTypes:  QualifiedName | 0..1 | The idref property specifies a globally unique identifier for an Object specified elsewhere. When the idref property is used, the id property MUST NOT also be specified and the other properties of the ObjectType class SHOULD NOT hold any content. |
| **has\_changed** | basicDataTypes:  Boolean | 0..1 | The has\_changed property specifies whether the Object has changed in some way. This property can be leveraged within a pattern Observable, triggering on whether the value of an Object specification has changed. This property is NOT intended to be used for versioning of CybOX content. |
| **State** | cyboxCommon:  VocabularyStringType | 0..1 | The State property specifies the state of the Object. Examples of potential states include *exists, inactive*, and *locked* (these specific values are only provided to help explain the property: they are neither recommended values nor necessarily part of any existing vocabulary). The content creator may choose any arbitrary value or may constrain the set of possible values by referencing an externally-defined vocabulary or leveraging a formally defined vocabulary extending from the cyboxCommon:ControlledVocabularyStringType class. The CybOX default vocabulary class for use in the property is ‘*ObjectStateVocab-1.0*’. |
| **Description** | cyboxCommon:  StructuredTextType | 0..1 | The Description property captures a textual description of the Object. Any length is permitted. Optional formatting is supported via the structuring\_format property of the StructuredTextType class. |
| **Properties** | cyboxCommon:  ObjectPropertiesType | 0..1 | The Properties property captures Object properties. CybOX defines a broad collection of Object data models that can be used to extend the cyboxCommon:ObjectPropertiesType superclass. Examples include the File Object data model, the Address Object data model, and the Network Packet data model. |
| **Domain\_Specific\_**  **Object\_Properties** | DomainSpecific  ObjectPropertiesType | 0..1 | The Domain\_Specific\_Object\_Properties property captures domain specific Object properties (i.e., metadata). |
| **Location** | cyboxCommon:  LocationType | 0..1 | The Location property characterizes the actual physical location of the Object. A simple location name may be specified or the underlying abstract class may be extended, in which case, the default and strongly RECOMMENDED subclass is CIQAddress3.0InstanceType, as defined in *[CybOX Version 2.2.1 Part XXX: Default Extensions](#AdditionalArtifacts).* |
| **Related\_Objects** | RelatedObjectsType | 0..1 | The Related\_Objects property specifies a set of one or more Objects related to this Object. |
| **Defined\_Effect** | DefinedEffectType | 0..1 | The Defined\_Effect property characterizes the effect that an Action has on an Object. Examples include *data read* and *state change*. |
| **Discovery\_Method** | cyboxCommon:  MeasureSourceType | 0..1 | The Discovery\_Method property characterizes how the Object was observed (in the case of a cyber observable Object instance) or could potentially be observed (in the case of a cyber observable Object pattern). Examples of details captured include identifying characteristics, time-related attributes, and a list of the tools used to collect the information. |

### ObservableType Class

The ObservableType class characterizes a cyber Observable. As shown in **Figure 3‑1**, a CybOX Observable pattern can either be on a CybOX Object with type corresponding to the CybOX ObjectType class (e.g., a File with name X), a CybOX Event with type corresponding to the CybOX EventType class (an Event is typically one or more actions taken against one or more Objects; e.g., “delete the File with name X”), or an Observable Composition with type corresponding to the CybOX ObservableCompositionType class.

[NEED DIAGRAM]

Figure ‑. UML diagram of the ObservableType class

The property table given in **Table 3‑4** corresponds to the UML diagram shown in **Figure 3‑1**.

Table ‑. Properties of the ObservableType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **id** | basicDataTypes:  QualifiedName | 0..1 | The id property specifies a globally unique identifier for the Observable. |
| **idref** | basicDataTypes:  QualifiedName | 0..1 | The idref property specifies a globally unique identifier for an Observable specified elsewhere. When the idref property is used, the id property MUST NOT also be specified and the other properties of the ObservableType class SHOULD NOT hold any content. |
| **negate** | basicDataTypes:Boolean | 0..1 | The negate property, when set to true, specifies the absence (rather than the presence) of the Observable in a CybOX pattern. |
| **sighting\_count** | basicDataTypes:  PositiveInteger | 0..1 | The sighting\_count property specifies how many different (but identical) instances of the Observable have been observed. |
| **Title** | basicDataTypes:BasicString | 0..1 | The Title property captures a title for the Observable and reflects what the content producer thinks the Observable as a whole should be called. The Title property is typically used by humans to reference a particular Observable; however, it is not suggested for correlation. |
| **Description** | cyboxCommon:  StructuredTextType | 0..1 | The Description property captures a textual description of the Observable. Any length is permitted. Optional formatting is supported via the structuring\_format property of the StructuredTextType class. |
| **Keywords** | KeywordsType | 0..1 | The Keywords property captures relevant keywords for the Observable. |
| **Observable\_Source** | cyboxCommon:  MeasureSourceType | 0..\* | 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. |
| **Object** | ObjectType | 0..1 | The Object property characterizes a specific cyber-relevant Object (e.g. a file, a registry key or a process). |
| **Event** | EventType | 0..1 | The Event property characterizes an observable Event relevant to the Observable. |
| **Observable\_Composition** | ObservableCompositionType | 0..1 | The Observable\_Composition property characterizes a composite Observable pattern expression. |
| **Pattern\_Fidelity** | PatternFidelityType | 0..1 | The Pattern\_Fidelity property characterizes the fidelity of the Observable pattern. |

## Secondary Classes

These classes are shared by CybOX classes.

### ActionArgumentType Class

The ActionArgumentType class characterizes an argument or parameter relevant to an Action.

Table ‑. Properties of the ActionArgumentType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Argument\_Name** | cyboxCommon:  VocabularyStringType | 0..1 | The Argument\_Name property specifies the name of the argument. Examples of potential names include *application name, base address,* and *size* (these specific values are only provided to help explain the property: they are neither recommended values nor necessarily part of any existing vocabulary). The content creator may choose any arbitrary value or may constrain the set of possible values by referencing an externally-defined vocabulary or leveraging a formally defined vocabulary extending from the cyboxCommon:ControlledVocabularyStringType class. The CybOX default vocabulary class for use in the property is ‘*ActionArgumentNameVocab-1.0*’. |
| **Argument\_Value** | basicDataTypes:  BasicString | 0..1 | The Argument\_Value property specifies the value of the Action argument or parameter. |

### ActionPertinentObjectPropertyType Class

The ActionPertinentObjectPropertyType class characterizes a property of an Object that is relevant to an Action.

Table ‑. Properties of the ActionPertinentObjectPropertyType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **name** | basicDataTypes:BasicString | 0..1 | The name property specifies the field name for the Object property. |
| **xpath** | basicDataTypes:BasicString | 0..1 | The xpath property specifies the XPath 1.0[[1]](#endnote-1) expression identifying the pertinent property of the data model that corresponds to the Object’s class. |

### ActionRelationshipType Class

The ActionRelationshipType class characterizes a relationship between one Action and a related Action.

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

Table ‑. Properties of the ActionRelationshipType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Type** | cyboxCommon:  VocabularyStringType | 0..1 | The Type property specifies the type of relationship between two actions. Examples of potential types include *dependent on, preceded by*, and *equivalent to* (these specific values are only provided to help explain the property: they are neither recommended values nor necessarily part of any existing vocabulary). The content creator may choose any arbitrary value or may constrain the set of possible values by referencing an externally-defined vocabulary or leveraging a formally defined vocabulary extending from the cyboxCommon:ControlledVocabularyStringType class. The CybOX default vocabulary class for use in the property is ‘*ActionRelationshipTypeVocab-1.0*’. |
| **Action\_Reference** | ActionReferenceType | 1..\* | The Action\_Reference property captures a reference to the related Action. |

The ActionReferenceType class is defined because in some cases it is not appropriate to define an Action *only* in the context of another Action, and in those cases, an otherwise defined Action should be referenced.

#### ActionReferenceType Class

The ActionReferenceType class captures a reference to a related Action.

Table ‑. Properties of the ActionReferenceType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **action\_id** | basicDataTypes:QualifiedName | 0..1 | The action\_id property specifies a globally unique identifier of the Action referenced. |

### AssociatedObjectType Class

The AssociatedObjectType class characterizes a cyber observable Object associated with a given cyber observable Action (i.e., an Object that is initiated by or affected by the Action). It extends the ObjectType superclass (see Section **3.1.3**).

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

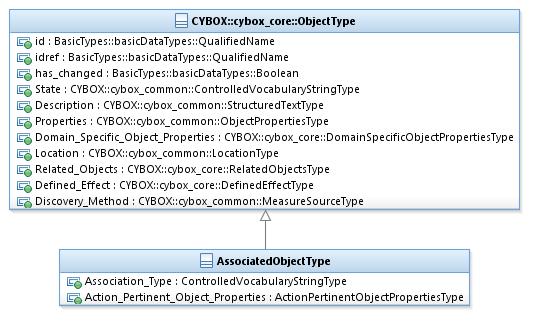


Figure ‑. UML diagram of the AssociatedObjectType class

The property table given in **Table 3‑9** corresponds to the UML diagram shown in **Figure 3‑2**.

Table ‑. Properties of the AssociatedObjectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Association\_Type** | cyboxCommon:  VocabularyStringType | 0..1 | The Association\_Type property specifies types of Action-Object associations. Examples of potential types include *initiating, affected*, and *utilized* (these specific values are only provided to help explain the property: they are neither recommended values nor necessarily part of any existing vocabulary). The content creator may choose any arbitrary value or may constrain the set of possible values by referencing an externally-defined vocabulary or leveraging a formally defined vocabulary extending from the cyboxCommon:ControlledVocabularyStringType class. The CybOX default vocabulary class for use in the property is ‘*ActionObjectAssociationTypeVocab-1.0’*. |
| **Action\_Pertinent\_**  **Object\_Properties** | ActionPertinent  ObjectPropertiesType | 0..1 | The Action\_Pertinent\_Object\_Properties property specifies a set of one or more properties of the Object that are pertinent to the Action. |

### DomainSpecificObjectPropertiesType Class

The DomainSpecificObjectPropertiesType class captures domain-specific metadata for a cyber Object. The DomainSpecificObjectPropertiesType class is an abstract class, and it MUST be extended via a subclass to specify the metadata. It has no properties.

Example domains that may be used to specialize the DomainSpecificObjectPropertiesType class include malware analysis and forensics.

### FrequencyType Class

The FrequencyType class characterizes the frequency of a given Action or Event.

Table ‑. Properties of the FrequencyType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **rate** | xs:float | 0..1 | The rate property specifies the rate of the frequency. |
| **units** | basicDataTypes:BasicString | 0..1 | The units property specifies the units of the frequency. |
| **scale** | basicDataTypes:BasicString | 0..1 | The scale property specifies the time scale of the frequency. |
| **trend** | TrendEnum | 0..1 | The trend property specifies the trend of the frequency. This property could be used as a trigger within an Event or Action pattern observable. |

### ObfuscationTechniqueType Class

The ObfuscationTechniqueType class characterizes a technique an attacker could potentially leverage to obfuscate the observability of the Observable.

Table ‑. Properties of the ObfuscationTechniqueType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Description** | cyboxCommon:  StructuredTextType | 1 | The Description property captures a textual description of the obfuscation technique. Any length is permitted. Optional formatting is supported via the structuring\_format property of the StructuredTextType class. |
| **Observables** | ObservablesType | 0..1 | The Observables property captures potential cyber Observables that could indicate the use of the obfuscation technique. |

### ObservableCompositionType Class

The ObservableCompositionType class enables a content creator to define a composite Observable pattern expression through the specification of a single Boolean operator (the operator of the expression) and a list of simple (non-composition) Observable patterns (the operands of the expression).

More complex Observable compositions (of the ObservableCompositionType class) can be created using multiple simple Observable patterns and/or other Observable compositions. For example, it may be desired to express the Observable Composition, OCX = OA AND (EA OR EB). The Observable (OA) and each of the two Events (EA and EB) would be created individually, and then one Observable Composition would be defined as OCA = EA OR EB. This permits OCX to be rewritten as OCX = OA AND OCA. Note that this example shows just one or several possible constructions that generate OCX.

Table ‑. Properties of the ObservableCompositionType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **operator** | OperatorTypeEnum | 0..1 | The operator property specifies the logical operator of the Observable. |
| **Observable** | ObservableType | 0..\* | The Observable property characterizes a cyber Observable. |

### PatternFidelityType Class

The PatternFidelityType class characterizes the fidelity of an Observable pattern.

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

Table ‑. Properties of the PatternFidelityType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Noisiness** | NoisinessEnum | 0..1 | The Noisiness property specifies the noisiness of the Observable (i.e., the false positives of the Observable). |
| **Ease\_of\_Evasion** | EaseOfObfuscationEnum | 0..1 | The Ease\_of\_Evasion property specifies the difficulty a threat actor may have obfuscating the Observable. |
| **Evasion\_Techniques** | ObfuscationTechniquesType | 0..1 | The Evasion\_Techniques property specifies a set of one or more potential techniques an attacker could leverage to obfuscate an Observable. |

### RelatedObjectType Class

The RelatedObjectType class characterizes a relationship between one Object and a related Object. It extends the ObjectType superclass by specifying the type of the relationship.

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

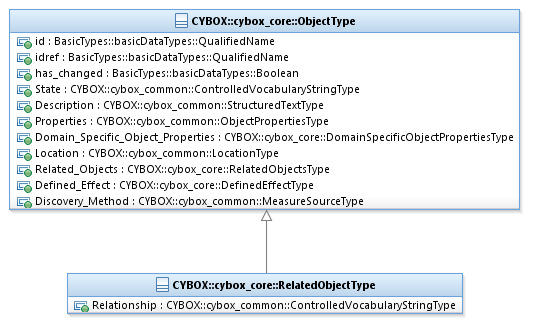


Figure ‑. UML diagram of the RelatedObjectType class

The property table given in **Table 3‑14** corresponds to the UML diagram shown in **Figure 3‑3**.

Table ‑. Properties of the RelatedObjectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Relationship** | cyboxCommon:  VocabularyStringType | 0..1 | The Relationship property specifies the type of relationship between two Objects. Examples of potential types include *created by, deleted by,* and *read from* (these specific values are only provided to help explain the property: they are neither recommended values nor necessarily part of any existing vocabulary). The content creator may choose any arbitrary value or may constrain the set of possible values by referencing an externally-defined vocabulary or leveraging a formally defined vocabulary extending from the cyboxCommon:ControlledVocabularyStringType class. The CybOX default vocabulary class for use in the property is ‘*ObjectRelationshipVocab-1.1*’. |

## Content Aggregation Classes

A content aggregation class captures a collection of one or more CybOX objects.

### ActionAliasesType Class

The ActionAliasesType class specifies a set of one or more names of an Action.

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

Table ‑. Properties of the ActionAliasesType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Action\_Alias** | basicDataTypes:BasicString | 1..\* | The Action\_Alias property specifies an alias name for the Action. |

### ActionArgumentsType Class

The ActionArgumentsType class specifies a set of one or more arguments or parameters relevant to an Action.

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

Table ‑. Properties of the ActionArgumentsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Action\_Argument** | ActionArgumentType | 1..\* | The Action\_Argument property characterizes an argument or parameter relevant to the Action. |

### ActionPertinentObjectPropertiesType Class

The ActionPertinentObjectPropertiesType class specifies a set of one or more Object properties pertinent to an Action.

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

Table ‑. Properties of the ActionPertinentObjectPropertiesType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Property** | ActionPertinentObjectPropertyType | 1..\* | The Property property characterizes an Object property pertinent to the Action. |

### ActionRelationshipsType Class

The ActionRelationshipsType specifies a set of one or more relationships between one Action and other Actions.

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

Table ‑. Properties of the ActionRelationshipsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Relationship** | ActionRelationshipType | 1..\* | The Relationship property characterizes a relationship between the Action and another, related Action. |

### ActionsType Class

The ActionsType class specifies a set of one or more cyber observable Actions.

The properties of the ActionsType class are given in **Table 3‑19**.

Table ‑. Properties of the ActionsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Action** | ActionType | 1..\* | The Action property characterizes a cyber observable Action. |

### AssociatedObjectsType Class

The AssociatedObjectsType class specifies a set of one or more cyber Objects relevant to an Action.

The properties of the AssociatedObjectsType class are given in **Table 3‑20**.

Table ‑. Properties of the AssociatedObjectsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Associated\_Object** | AssociatedObjectType | 1..\* | The Associated\_Object property characterizes a cyber Object associated with the Action. An associated Object may be an Object that initiated the Action, an Object that is affected by or utilized by the Action, or an Object that is returned as a result of the Action. |

### KeywordsType Class

The KeywordsType class specifies a set of one or more keywords.

The properties of the KeywordsType class are given in **Table 3‑21**.

Table ‑. Properties of the KeywordsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Keyword** | basicDataTypes:BasicString | 1..\* | The Keyword property captures a keyword. |

### ObfuscationTechniquesType Class

The ObfuscationTechniquesType class specifies a set of one or more potential techniques an attacker could leverage to obfuscate an Observable.

The properties of the AssociatedObjectsType class are given in **Table 3‑22**.

Table ‑. Properties of the ObfuscationTechniquesType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Obfuscation\_Technique** | ObfuscationTechniqueType | 1..\* | The Obfuscation\_Technique property characterizes a technique an attacker could potentially leverage to obfuscate the Observable. |

### ObservablesType Class

The ObservablesType class characterizes a set of one or more cyber Observables.

The properties of the ObservablesType class are given in **Table 3‑23**.

Table ‑. Properties of the ObservablesType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **cybox\_major\_version** | basicDataTypes:  BasicString | 0..1 | The cybox\_major\_version property specifies the major version of the CybOX language used for the set of Observables. |
| **cybox\_minor\_version** | basicDataTypes:  BasicString | 0..1 | The cybox\_minor\_version property specifies the minor version of the CybOX language used for the set of Observables. |
| **cybox\_update\_version** | basicDataTypes:  BasicString | 0..1 | The cybox\_update\_version property specifies the update version of the CybOX language used for this set of Observables. This property MUST be used when using an update version of CybOX. |
| **Observable\_Package\_Source** | cyboxCommon:  MeasureSourceType | 0..1 | The Observable\_Package\_Source property characterizes the source of the Observables information. Examples of details captured include identifying characteristics, time-related attributes, and a list of the tools used to collect the information. |
| **Observable** | ObservableType | 1..\* | The Observable property characterizes a cyber Observable. |
| **Pools** | PoolsType | 0..1 | The Pools property captures the Events, Actions, Objects and Properties (in a space-efficient, pooled manner) that are referenced by the cyber Observable. |

### PropertiesType Class

The PropertiesType class specifies a set of one or more properties enumerated as a result of the Action on the Object.

The properties of the PropertiesType class are given in **Table 3‑24**.

Table ‑. Properties of the PropertiesType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Property** | basicDataTypes:  BasicString | 1..\* | The Property property specifies a property resulting from an Action on an Object. |

### RelatedObjectsType Class

The RelatedObjectsType class specifies a set of one or more relationships between one Object and other Objects.

The properties of the RelatedObjectsType class are given in **Table 3‑25**.

Table ‑. Properties of the RelatedObjectsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Related\_Object** | RelatedObjectType | 1..\* | The Related\_Object property specifies an Object related to the Object of focus and characterizes the relationship between the Objects. |

### ValuesType Class

The ValuesType class specifies a set of one or more values that are enumerated as a result of an Action on an Object.

The properties of the ValuesType class are given in **Table 3‑26**.

Table ‑. Properties of the ValuesType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Value** | basicDataTypes:BasicString | 1..\* | The Value property specifies a single value that is enumerated as a result of the Action on the Object. |

## Pool Classes

Pool classes enable observable elements – Events, Actions, Objects, and Properties – to be described in a space-efficient manner. Rather than defining identical observable elements multiple times within a set of defined Observables, observable elements are defined in type-specific pools (i.e., sets) and are then referenced by Observable structures.

### PoolsType Class

The PoolsType class captures one or more pools, each of which contains one type of observable element.

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



Figure ‑. UML diagram of the PoolsType class

The property table given in **Table 3‑27** corresponds to the UML diagram shown in **Figure 3‑4**.

Table ‑. Properties of the PoolsType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Event\_Pool** | EventPoolType | 0..1 | The Event\_Pool property specifies a pool of one or more Events. |
| **Action\_Pool** | ActionPoolType | 0..1 | The Action\_Pool property specifies a pool of one or more Actions. |
| **Object\_Pool** | ObjectPoolType | 0..1 | The Object\_Pool property specifies a pool of one or more Objects. |
| **Property\_Pool** | PropertyPoolType | 0..1 | The Property\_Pool property specifies a pool of one or more Properties. |

### EventPoolType Class

The EventPoolType class specifies a pool of one or more Events.

The properties of the EventPoolType class are given in **Table 3‑28**.

Table ‑. Properties of the EventPoolType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Event** | EventType | 1..\* | The Event property characterizes a cyber observable Event. |

### ActionPoolType Class

The ActionPoolType class specifies a pool of one or more Actions.

The properties of the ActionPoolType class are given in **Table 3‑29.**

Table ‑. Properties of the ActionPoolType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Action** | ActionType | 1..\* | The Action property characterizes a cyber observable Action. |

### ObjectPoolType Class

The ObjectPoolType class specifies a pool of one or more Objects.

The properties of the ObjectPoolType class are given in **Table 3‑30**.

Table ‑. Properties of the ObjectPoolType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Object** | ObjectType | 1..\* | The Object property characterizes a cyber-relevant Object. |

### PropertyPoolType Class

The PropertyPoolType class specifies a pool of one or more Properties.

The properties of the PropertyPoolType class are given in **Table 3‑31**.

Table ‑. Properties of the PropertyPoolType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Property** | cyboxCommon:PropertyType | 1..\* | The Property property characterizes an Object property. |

## Defined Effect Classes

The CybOX Common data model defines a number of classes to characterize a broad range of potential effects of an Action on an Object. Characterization is achieved through specialization of the DefinedEffectType abstract class, which is defined in Section **3.5.1**. The defined effect-type classes that specialize the DefinedEffectType class are presented in Sections **3.5.2** through **3.5.10**, which corresponds to the order that they are listed in the EffectTypeEnum enumeration (Section **3.6.4**).

### DefinedEffectType Class

The DefinedEffectType class specifies the type of the effect that an Action has on an Object. It is an abstract class, and it MUST be extended via a subclass to specify a complete effect. Use of the DefinedEffectType class enables a broad range of complex Action effects on Objects to be specified.

A UML diagram corresponding to the DefinedEffectType class is shown in **Figure 3‑5** (due to space constraints, only three potential subclasses are shown, but the other subclasses presented in Sections Sections **3.5.5** through **3.5.10** could be included similarly).

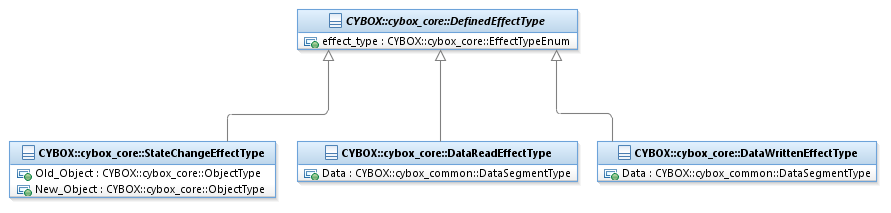


Figure ‑. UML diagram of the DefinedEffectType class

The property table given in **Table 3‑32** corresponds to the UML diagram shown in **Figure 3‑5**.

Table ‑. Properties of the DefinedEffectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **effect\_type** | EffectTypeEnum | 0..1 | The effect\_type property specifies the effect of the Action on the Object. |

### StateChangeEffectType Class

The StateChangeEffectType class extends the DefinedEffectType superclass by characterizing the effects of Actions upon Objects where the state of the Object is changed.

The properties of the StateChangeEffectType specialization are given in **Table 3‑33**.

Table ‑. Properties of the StateChangeEffectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Old\_Object** | ObjectType | 0..1 | The Old\_Object property characterizes the Object before the state change effect occurred. |
| **New\_Object** | ObjectType | 1 | The New\_Object property characterizes the Object after the state change effect occurred. |

### DataReadEffectType Class

The DataReadEffectType class extends the DefinedEffectType superclass by characterizing the effects of Actions upon Objects where some data is read, such as from a file or a pipe.

The properties of the DataReadEffectType specialization are given in **Table 3‑34**.

Table ‑. Properties of the DataReadEffectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Data** | cyboxCommon:DataSegmentType | 1 | The Data property characterizes the data that was read from the Object by the Action. |

### DataWrittenEffectType Class

The DataWrittenEffectType class extends the DefinedEffectType superclass by characterizing the effects of Actions upon Objects where some data is written, such as to a file or a pipe.

The properties of the DataWrittenEffectType specialization are given in **Table 3‑35**.

Table ‑. Properties of the DataWrittenEffectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Data** | cyboxCommon:DataSegmentType | 1 | The Data property characterizes the data that was written to the Object by the Action. |

### DataSentEffectType Class

The DataSentEffectType class extends the DefinedEffectType superclass by characterizing the effects of Actions upon Objects where some data is sent, such as a byte sequence on a socket.

The properties of the DataSentEffectType specialization are given in **Table 3‑36**.

Table ‑. Properties of the DataSentEffectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Data** | cyboxCommon:DataSegmentType | 1 | The Data property characterizes the data that was sent on the Object, or from the Object, by the Action. |

### DataReceivedEffectType Class

The DataReceivedEffectType class extends the DefinedEffectType superclass by characterizing the effects of Actions upon Objects where some data is received, such as a byte sequence on a socket.

The properties of the DataReceivedEffectType specialization are given in **Table 3‑37**.

Table ‑. Properties of the DataReceivedEffectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Data** | cyboxCommon:DataSegmentType | 1 | The Data property characterizes the data that was received on the Object, or from the Object, by the Action. |

### PropertyReadEffectType Class

The PropertyReadEffectType class extends the DefinedEffectType superclass by characterizing the effects of Actions upon Objects where some specific property is read from an Object, such as the current running state of a process.

The properties of the PropertyReadEffectType specialization are given in **Table 3‑38**.

Table ‑. Properties of the PropertyReadEffectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Name** | basicDataTypes:BasicString | 0..1 | The Name property specifies the name of the property being read. |
| **Value** | basicDataTypes:BasicString | 0..1 | The Value property specifies the value of the property being read. |

### PropertiesEnumeratedEffectType Class

The PropertiesEnumeratedEffectType class extends the DefinedEffectType superclass by characterizing the effects of Actions upon Objects where some properties of the Object are enumerated, such as the startup parameters for a process.

The properties of the PropertiesEnumeratedEffectType specialization are given in **Table 3‑39**.

Table ‑. Properties of the PropertiesEnumeratedEffectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Properties** | PropertiesType | 1 | The Properties property specifies a set of one or more values that are enumerated as a result of an Action on an Object. |

### ValuesEnumeratedEffectType Class

The ValuesEnumeratedEffectType class extends the DefinedEffectType superclass by characterizing the effects of Actions upon Objects where some values of the Object are enumerated, such as the values of a registry key.

The properties of the ValuesEnumeratedEffectType specialization are given in **Table 3‑40**.

Table ‑. Properties of the ValuesEnumeratedEffectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Values** | ValuesType | 1 | The Values property specifies a set of one or more values that are enumerated as a result of an Action on an Object. |

### SendControlCodeEffectType Class

The SendControlCodeEffectType class extends the DefinedEffectType superclass by characterizing the effects of Actions upon Objects where some control code, or other control-oriented communication signal, is sent to the Object.

The properties of the SendControlCodeEffectType specialization are given in **Table 3‑41**.

Table ‑. Properties of the SendControlCodeEffectType class

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Multiplicity** | **Description** |
| **Control\_Code** | basicDataTypes:BasicString | 1 | The Control\_Code property specifies the actual control code sent to the Object. |

## Enumerations

### ActionStatusTypeEnum Enumeration

The ActionStatusTypeEnum enumeration is an inventory of Action statuses. The enumeration literals are given in **Table 3‑42**.

Table ‑. Literals of the ActionStatusTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **Success** | The cyber observable Action was successful. |
| **Fail** | The cyber observable Action failed. |
| **Error** | The cyber observable Action resulted in an error. |
| **Complete/Finish** | The cyber observable Action completed or finished. This action status does not specify the result of the Action (e.g., Success, Error). |
| **Pending** | The cyber observable Action is pending. |
| **Ongoing** | The cyber observable Action is ongoing. |
| **Unknown** | The cyber observable Action has unknown status. |

### ActionContextTypeEnum Enumeration

The ActionContextTypeEnum enumeration is an inventory of Action contexts. The enumeration literals are given in **Table 3‑43**.

Table ‑. Literals of the ActionContextTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **Host** | The cyber observable Action occurred on a host. |
| **Network** | The cyber observable Action occurred on a network. |

### EaseOfObfuscationEnum Enumeration

The EaseOfObfuscationEnum enumeration is an inventory of values representing the difficulty a threat actor may have obfuscating an Observable. The enumeration literals are given in **Table 3‑44**.

Table ‑. Literals of the EaseOfObfuscationEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **High** | The Observable is very easy to obfuscate and hide. |
| **Medium** | The Observable is somewhat easy to obfuscate and hide. |
| **Low** | The Observable is not very easy to obfuscate and hide. |

### EffectTypeEnum Enumeration

The EffectTypeEnum enumeration is an inventory of values representing the effect of an Action on an Object. The enumeration literals are given in **Table 3‑45**.

Table ‑. Literals of the EffectTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **State\_Changed** | The Action changed the state of the Object. |
| **Data\_Read** | The Action read data from the Object. |
| **Data\_Written** | The Action wrote data to the Object. |
| **Data\_Sent** | The Action sent data to the Object. |
| **Data\_Received** | The Action received data from the Object. |
| **Properties\_Read** | The Action read properties of the Object. |
| **Properties\_Enumerated** | The Action enumerated properties of the Object. |
| **Values\_Enumerated** | The Action enumerated values of the Object. |
| **ControlCode\_Sent** | The Action sent control code to the Object. |

### NoisinessEnum Enumeration

The NoisinessEnum enumeration is an inventory of values for representing the noisiness of the Observable (i.e., false positives of the Observable). The enumeration literals are given in **Table 3‑46**.

Table ‑. Literals of the NoisinessEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **High** | The Observable has a high level of noisiness (i.e., a potentially high level of false positives). |
| **Medium** | The Observable has a medium level of noisiness (i.e., a potentially medium level of false positives). |
| **Low** | The Observable has a low level of noisiness (i.e., a potentially low level of false positives). |

### OperatorTypeEnum Enumeration

The OperatorTypeEnum enumeration is an inventory of valid operator types. The enumeration literals are given in **Table 3‑4**7.

Table ‑. Literals of the OperatorTypeEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| **AND** | Logical AND operator |
| **OR** | Logical OR operator |

### TrendEnum Enumeration

The TrendEnum enumeration is an inventory of valid trend types. The enumeration literals are given in **Table 3‑48**.

Table ‑. Literals of the TrendEnum enumeration

|  |  |
| --- | --- |
| **Enumeration Literal** | **Description** |
| Increasing | The trend is increasing. |
| **Decreasing** | The trend is decreasing. |

# # Conformance

The last numbered section in the specification must be the Conformance section. Conformance Statements/Clauses go here. [Remove # marker]

1. Acknowledgments

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

Participants:

[Participant Name, Affiliation | Individual Member]

[Participant Name, Affiliation | Individual Member]

1. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Editor** | **Changes Made** |
| wd01 | 28 August 2015 | Desiree Beck Trey Darley Ivan Kirillov Rich Piazza | Initial transfer to OASIS template |

1. XPath 1.0 is a language for selecting portions of XML documents. [↑](#endnote-ref-1)