CybOXTM Version 2.1.1 XML Binding Specification Version 1.0

Working Draft 01

13 May 2016

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/)

Additional artifacts:

This prose specification is one component of a Work Product that also includes:

* XML schemas: (list file names or directory name)
* Other parts (list titles and/or file names)

Related work:

This specification is related to:

* *CybOX Version 2.1.1* Work in progress.

Declared XML namespaces:

* list namespaces declared within this specification

Abstract:

This specification describes XML bindings for CybOX Version 2.1.1

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/cybox-2.1.1-xml-binding/v1.0/csd01/stix-1.2.1-xml-binding-v1.0-csd01.docx

Permanent “Latest version” URI:  
http://docs.oasis-open.org/cti/stix-1.2.1-xml-binding/v1.0/stix-1.2.1-xml-binding-v1.0.docx

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

Copyright © OASIS Open 2016. All Rights Reserved.

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

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

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

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

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

Table of Contents

[1 Introduction 4](#_Toc451328261)

[1.1 Document Conventions 4](#_Toc451328262)

[1.1.1 Fonts 4](#_Toc451328263)

[1.1.2 XML Namespaces 4](#_Toc451328264)

[1.1.3 UML Diagram Icons and Arrow Types 4](#_Toc451328265)

[1.1.4 XSD Examples 5](#_Toc451328266)

[1.2 Terminology 5](#_Toc451328267)

[1.3 Normative References 5](#_Toc451328268)

[2 Binding Rules 6](#_Toc451328269)

[2.1 UML Packages 6](#_Toc451328270)

[2.2 UML Classes 6](#_Toc451328271)

[2.3 UML Attributes and Associations 8](#_Toc451328272)

[2.4 UML Stereotypes 10](#_Toc451328273)

[2.5 UML Data Types 10](#_Toc451328274)

[2.6 UML Enumerations 12](#_Toc451328275)

[2.7 Controlled Vocabularies 13](#_Toc451328276)

[3 Relationships to the CybOX 2.1.1 XML Schemas 16](#_Toc451328277)

[3.1 UML Package to XML Namespace Name Mapping 16](#_Toc451328278)

[3.2 UML Abstract Classes 16](#_Toc451328279)

[3.3 Using XSD Data Types 17](#_Toc451328280)

[3.4 Extensions and Externally Defined Data Models 18](#_Toc451328281)

[3.5 XSD attributeGroups 19](#_Toc451328282)

[4 Conformance 22](#_Toc451328283)

[Appendix A. Acknowledgments 23](#_Toc451328284)

[Appendix B. Revision History 27](#_Toc451328285)

# Introduction

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 defines a binding of this implementation-agnostic specification to an XML implementation, by describing binding rules to convert the UML formalism to a series of XML schemas – one for each data model that make up the CybOX UML model.

In Section **2**, we define a binding rule for each concept in UML that was used in the CybOX specification. Because the CybOX XML schema were developed before the UML model, Section **3** is used to describe design choices that impact the UML model and have implications for the binding rules, especially exceptions to those rules that can be found in the official CybOX XML [**schemas**](#additional_artifacts). Conformance information can be found in Section **4.**

This document identifies the XML schemas that are part of this OASIS product and describes how the UML models were generated for CybOX. The XML schemas were developed previously to the UML model and are officially defined as the normative XML schema definition for CybOX 2.1.1.  The rest of this document is informative only – as it describes a possible process to create the XSD models from the UML models. As the text below is informative, any words used that are coincidently defined in [**RFC2119**](#rfc2119) should be read as only having their usual English meaning.

## 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.

Examples: Event, Object, Action

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

Examples: xs:complexType, xs:string, AddressObj:AddressObjectType

### XML Namespaces

Each CybOX data model is captured in a different XML schema, related to the UML packages which together compose the full CybOX UML model. The CybOX XSD specification has namespaces which correspond to the UML packages. To refer to a particular type of a specific XML schema, we use the format namespace:type, where namespace corresponds to the appropriate XML namespace. Likewise, we use the format package-name:class for UML classes.

### UML Diagram Icons and Arrow Types

Diagram icons are used in a UML diagram to indicate whether a shape is a class, enumeration or 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** |
|  | 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. |

### XSD Examples

To improve readability, some XML in examples was altered from the actual XML schemas.

## 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](#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>.

# Binding Rules

## UML Packages

A UML *package* is a concept that is useful for reducing the complexity of a data model by focusing on one aspect at a time. In addition, it allows for the scoping of the names used. The concept of a namespace in XSD is similar. Each of the eighty-eight CybOX objects that have been define is represented using an individual UML package. Other UML packages correspond to the overarching data models in CybOX, such as Core and Common. The use of multiple packages allows the CybOX data model to be modular: all of the CybOX components are defined in separate packages rather than in one large package to limit interdependence between CybOX components. In XML, this corresponds to separate XML schemas, each with their own namespace.

**Binding Rule 1:** The XML binding for a UML Package is an XSD schema with its own namespace.

* Each package should correspond to a separate XSD schema in a separate file.
* The mapping from namespace name to package name is found in Section 3.1.
* The target namespace of that schema must be declared as follows:
  + targetNamespace=http://docs.oasis-open.org/cti/ns/cybox*/<namespace-name>*-2
* The prefix name must be declared. The package name and prefix name are the same.
  + xmlns:<*namespace-prefix*>=

"http://docs.oasis-open.org/cti/ns/cybox/<*namespace-name*>-2”

* Other XML schemas of the CybOX model must be imported as needed, using the xs:import construct. The namespace prefix of any imported schema must also be specified.

## UML Classes

A *class* in UML corresponds to xs:complexType in XSD. In UML, classes are related to each other using *generalization*, which indicates that one is derived from the other. In XSD, generalization corresponds to an *extension* of the type.

**Binding Rule 2:** The XML binding for a UML class is an XSD complex type.

* The names of an XSD complex type in the XML schema must have the same name as the corresponding class in the UML model. Specifically, both have the “Type” suffix.
* In both formalisms, the classes/types are related from the more specific to the more general. If the class has a generalization, then the corresponding complex type is specified using the xs:extension construct.
* Complex types that are extensions of other complex types must use the xs:complexContent construct, as shown in the example below.
* Certain XSD complex types should be declared as abstract. See Section **3.2** for guidance.
* All xs:element construct must be enclosed within the xs:sequence construct.

Here is a generalization relationship from the UML model:

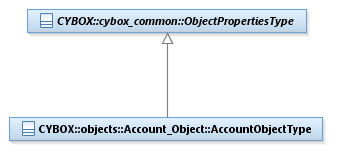


Figure ‑. UML Generalization

Here is the equivalent definition from the XSD:

<xs:complexType name="AccountObjectType">

<xs:complexContent>

<xs:extension base="cyboxCommon:ObjectPropertiesType">

<!-- snip! -->

</xs:complexContent>

</xs:complexType>

XSD simple types are used to define data types and enumerations. Data types and enumerations are discussed in Sections **2.4** and **2.5**, respectively.

## UML Attributes and Associations

UML *attributes* are associated with UML classes. A UML attribute A can be thought of as a *property* of UML class B. A UML attribute has various facets:

* Type - can be a UML data type or a UML class
* Multiplicity - indicates how many objects are allowed in the attribute.
* Aggregation – indicates whether the object is a part of (or owned by) another object, or whether it can exist in various contexts[[1]](#endnote-1).

**Binding Rule 3:** The XML binding for a UML attribute is an XSD element or an XSD attribute.

* The capitalization of the UML attribute’s name determines whether an XSD attribute or XSD element is used. The name itself must remain the same, including capitalization.
* The type of the XSD element is either the XSD complex type that corresponds to the UML class of the UML attribute or an XSD simple type that corresponds to a UML data type of the UML attribute. The correspondences of UML data types to predefined XSD data types from the xs namespace is given in **Table 3‑2**.
* If the type of the UML attribute is cyboxCommon:VocabularyStringType, then the type of the corresponding XSD element must be cybox::Common:ControlledVocabularyStringType.
* Order of the elements within the xs:sequence tags is important, and must conform to the order as they appear in the [**specification document**](#related_work) tables.
* The multiplicity of an XSD attribute is implicitly always 0..1.
* The multiplicity of an XSD element is derived using the mapping in **Table 2‑1**. Because of default values, either minOccurs or maxOccurs can be omitted (as indicated by italics).
* Some UML attributes define a default value. That value should be specified using the default XSD attribute “default”.

Table ‑: Multiplicity Mapping

|  |  |
| --- | --- |
| **UML Multiplicity** | **XSD minOccurs and maxOccurs** |
| 1 | minOccurs=”1”  *maxOccurs=”1”* |
| 0..\* | minOccurs=”0”  maxOccurs=”unbounded” |
| 0..1 | minOccurs=”0”  *maxOccurs=”1”* |
| 1..\* | *minOccurs=”1”*  maxOccurs=”unbounded” |

Examples of UML attributes of a UML class are shown in the following diagram:

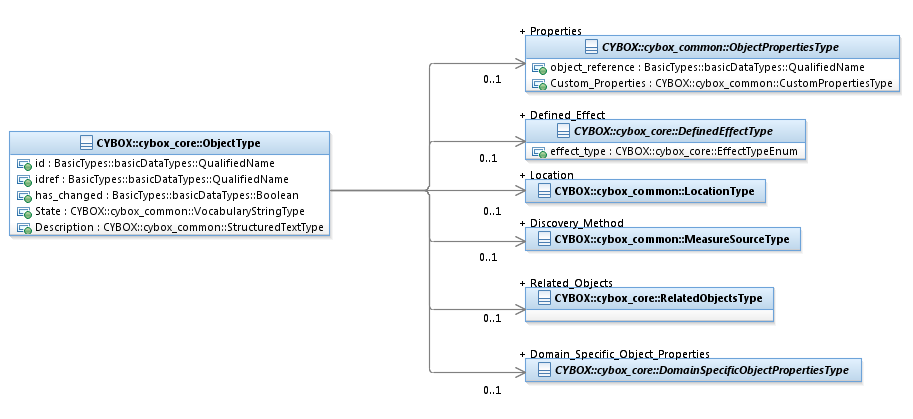


Figure ‑. UML ObjectType class

Here is the equivalent definition from the XSD:

<xs:complexType name="ObjectType">

<xs:sequence>

<xs:element name="State"

type="cyboxCommon:ControlledVocabularyStringType"

minOccurs="0" maxOccurs="1">

</xs:element>

<xs:element name="Description"

type="cyboxCommon:StructuredTextType" minOccurs="0">

</xs:element>

<xs:element name="Properties"

type="cyboxCommon:ObjectPropertiesType" minOccurs="0">

</xs:element>

<xs:element name="Domain\_Specific\_Object\_Properties"

type="cybox:DomainSpecificObjectPropertiesType"

minOccurs="0">

</xs:element>

<xs:element name="Location"

type="cyboxCommon:LocationType" minOccurs="0">

</xs:element>

<xs:element name="Related\_Objects"

type="cybox:RelatedObjectsType" minOccurs="0">

</xs:element>

<xs:element name="Defined\_Effect"

type="cybox:DefinedEffectType" minOccurs="0">

</xs:element>

<xs:element name="Discovery\_Method"

type="cyboxCommon:MeasureSourceType" minOccurs="0">

</xs:element>

</xs:sequence>

<xs:attribute name="id" type="xs:QName"/>

<xs:attribute name="idref" type="xs:QName"/>

<xs:attribute name="has\_changed" type="xs:boolean"/>

</xs:complexType>

## UML Stereotypes

<<choice>> was the one UML stereotype that was created for the CybOX 2.1.1 UML model. This is to specify a concept similar to specialization in UML, without introducing UML interfaces or anonymous UML classes. Anonymous UML classes are used when the class will not be directly referred to elsewhere in the UML model, therefore no name is necessary. The choice stereotype corresponds to the xs:choice construct in XSD.



Figure ‑. UML Stereotype

**Binding Rule 4:**  The XML binding for a UML class that is associated with the <<choice>> UML stereotype, called the choice class, correspond to the xs:choice construct in XSD.

* All such classes have an association named “Has\_Choice” from another UML class in the model, called the parent class.
* The UML attributes of the choice type become xs:elements in the parent class, surrounded by the xs:choice construct. The xs:choice construct replaces the “Has\_Choice” property, and must be placed in the order among the xs:elements where the “Has\_Choice” property appears in the [**specification document**](#related_work) tables, unless “Has\_Choice” is the only property.
* The choice class does not need to be bound to any additional XSD syntax.

Here is the equivalent definition from the XSD:

<xs:complexType name="SocketAddressObjectType">

<xs:complexContent>

<xs:extension base="cyboxCommon:ObjectPropertiesType">

<xs:sequence>

<xs:choice>

<xs:element name="IP\_Address"

type="AddressObj:AddressObjectType"

minOccurs="0"/>

<xs:element name="Hostname"

type="HostnameObj:HostnameObjectType"

minOccurs="0"/>

</xs:choice>

<xs:element name="Port" type="PortObj:PortObjectType"

minOccurs="0"/>

</xs:sequence>

</xs:extension>

</xs:complexContent>

</xs:complexType>

## UML Data Types

As stated in the UML 2.4.1 specification (ref), UML *data types* are similar to UML classes, but have an important difference:

“A data type is a special kind of classifier, similar to a class. It differs from a class in that instances of a data type are identified only by their value*. All copies of an instance of a data type and any instances of that data type with the same value are considered to be equal instances* (ed. emphasis added). Instances of a data type that have attributes (i.e., is a structured data type) are considered to be equal if the structure is the same and the values of the corresponding attributes are equal. If a data type has attributes, then instances of that data type will contain attribute values matching the attributes.”

Because some of the basic UML data types are used in various data models in addition to CybOX, we introduced a separate UML package, basicDataTypes, to hold most of the basic UML data types. See **Table 3‑2**.

Other UML Data Types are used to define properties that can contain either an enumeration literal or a simple string.

**Binding Rule 5**: The XML binding for a UML data type corresponds to an XSD simple type.

* For the common basic data types defined in the UML model, there exists a corresponding predefined XSD simple type defined in the xs namespace. Because these are predefined, they need to be explicitly defined in the implementation, except for the use of the xs namespace.
* For UML data types that are associated with a corresponding UML enumeration (i.e., they have the same name except the enumeration’s name has an “Enum” suffix), the following template is used to create the XSD complex type:

<xs:complexType name=”*data type name”*

<xs:simpleContent>

<xs:restriction base="cyboxCommon:BaseObjectPropertyType">

<xs:simpleType>

<xs:union memberTypes=”*data type name*Enum,

xs:string”/>

<xs:attribute name="datatype"

type="cyboxCommon:DatatypeEnum"

fixed="string">

</xs:attribute>

</xs:restriction>

</xs:simpleContent>

</xs:complexType>

* For UML data types that are *not* associated with a corresponding UML enumeration, the template is the same, except the xs:union contains only xs:string



Figure ‑. UML Data Types

An example of using this template:

The DiskObject package contains a UML data type named DiskType, and a corresponding UML enumeration named DiskTypeEnum. The existence of these two UML artifacts is represented in the XML schema using the following XSD complex type.

<xs:complexType name="DiskType">

<xs:simpleContent>

<xs:restriction base="cyboxCommon:BaseObjectPropertyType">

<xs:simpleType>

<xs:union memberTypes="DiskObj:DiskTypeEnum

xs:string"/>

</xs:simpleType>

<xs:attribute name="datatype"

type="cyboxCommon:DatatypeEnum"

fixed="string">

</xs:attribute>

</xs:restriction>

</xs:simpleContent>

</xs:complexType>

The XSD syntax for the UML enumeration is covered in the next section.

The BaseObjectPropertyGroup and PatternFieldGroup UML data types are special cases, which are covered in Section 3.5.

## UML Enumerations

UML *enumerations* are extensions to the concept of UML data types. An enumeration defines a complete collection of user-defined *literals* that are members of a set with particular semantics (e.g., type of disks).

**Binding Rule 6:** The XML binding for a UML enumeration is an XSD simple type using the xs:enumeration construct.

* *This binding rule only applies to UML enumerations not used to model CybOX Controlled Vocabularies*.
* Introduce a XSD simple type, whose base type is an xs:restriction of xs:string. For each UML enumeration literal introduce an xs:enumeration whose value attribute corresponds to the name of the UML enumeration literal.

The following UML diagram specifies a UML enumeration:

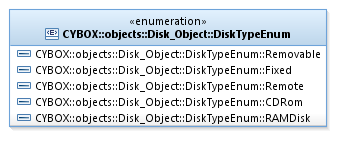


Figure ‑. UML Enumeration

This would be represented in XSD as:

<xs:simpleType name="DiskTypeEnum">

<xs:restriction base="xs:string">

<xs:enumeration value="Removable"/>

<xs:enumeration value="Fixed">

<xs:enumeration value="Remote">

<xs:enumeration value="CDRom">

<xs:enumeration value="RAMDisk">

</xs:restriction>

</xs:simpleType>

Most enumerations used in CybOX are for default vocabularies. In the Section **2.7,** we provide a full description of the XML binding of default vocabularies.

## Controlled Vocabularies

The binding rule for the UML model of CybOX Controlled Vocabularies is significantly more complex from the previously described rules. The binding rule produces a semantically equivalent XSD model, but syntactically, it differs significantly. This is caused by certain mismatches of the semantics of UML classes, UML data types and UML enumerations and XSD complex types, XSD simple types and XSD enumerations.



Figure ‑. CybOX Controlled Vocabulary UML Model

The specification for controlled vocabularies is as follows, and all data model implementations must allow for all of these use cases:

1. Leverage a formally defined default vocabulary. A collection of default vocabularies and associated enumerations that are based on input from the CybOX community.
2. Formally define and leverage a custom vocabulary. Producers and consumers agree upon a common vocabulary that they use in the sharing of CybOX documents.
3. Reference an externally-defined, custom vocabulary. Externally-defined vocabularies that have been explicitly defined by standards organizations.
4. Choose an arbitrary and unconstrained value. A free-form string.
5. Use a pattern

**Binding Rule 7a:** Introduce an XSD complex type cyboxCommon:ControlledVocabularyStringType, to encapsulate all of the use cases supported by UML data types cyboxCommon:PatternFieldGroup, cyboxCommon:VocabularyStringType, cyboxCommon:UnenforcedVocabularyStringType and cyboxCommon:ControlledVocabularyStringType. Add XSD attributes for vocab\_name and vocab\_references as shown in the example.

**BindingRule 7b:** For each UML enumeration defined in the UML package cyboxVocabs:

* Introduce one xs:simpleType, which is a XSD restriction of xs:string, and contains an enumeration element for each enumeration literal from the UML enumeration using the xs:enumeration construct. The name of this type is formed by replacing “Vocab” in the UML enumeration name by “Enum”.
* Introduce one xs:complexType of the same name as the UML enumeration. Use the xs:simpleContent tag, with a XSD *restriction* of the XSD complex type cyboxCommon:ControlledVocabularyStringType
  + Within the XSD restriction add an XSD simple type, that includes using the xs:union construct to include the Enum type introduced previously.
  + For documentation purposes, you can add vocab\_name and vocab\_reference attributes using the fixed construct.

Here is the XSD for defining controlled vocabularies, using ActionRelationshipTypeVocab as the exemplar.

<xs:complexType name="ControlledVocabularyStringType">

<xs:simpleContent>

<xs:extension base="xs:anySimpleType">

<xs:attribute name="vocab\_name" type="xs:string" use="optional"/>

<xs:attribute name="vocab\_reference" type="xs:anyURI"

use="optional"/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

If an XSD complex type has an XSD element that specifies some controlled vocabulary it should use cyboxVocabs:ControlledVocabularyStringType as its type. This allows contributors of CybOX content to be able to specify values for a controlled vocabulary that satisfy all use cases from the above specification.

<xs:simpleType name="ActionRelationshipTypeEnum-1.0">

<xs:restriction base="xs:string">

<xs:enumeration value="Preceded\_By"/>

<xs:enumeration value="Followed\_By"/>

<xs:enumeration value="Equivalent\_To"/>

<xs:enumeration value="Related\_To"/>

<xs:enumeration value="Dependent\_On"/>

<xs:enumeration value="Initiated\_By"/>

<xs:enumeration value="Initiated"/>

</xs:restriction>

</xs:simpleType>

<xs:complexType name="ActionRelationshipTypeVocab-1.0">

<xs:simpleContent>

<xs:restriction base="cyboxCommon:ControlledVocabularyStringType">

<xs:simpleType>

<xs:union memberTypes="cyboxVocabs:ActionRelationshipTypeEnum-1.0"/>

</xs:simpleType>

<xs:attribute name="vocab\_name" type="xs:string" use="optional" fixed="CybOX Default Action-Action Relationships"/>

<xs:attribute name="vocab\_reference" type="xs:anyURI" use="optional" fixed=" http://docs.oasis-open.org/cti/cybox/v2.1.1/csd01/xml-schemas/default-vocabularies.xsd#ActionRelationshipTypeVocab-1.0"/>

</xs:restriction>

</xs:simpleContent>

</xs:complexType>

**Exception 1:** Most controlled vocabularies are defined in the cyboxVocabs package. Two exceptions are AuthenticationTokenProtectionMechanismTypeVocab-1.0 and AuthenticationTypeVocab-1.0 defined in the Account package.

# Relationships to the CybOX 2.1.1 XML Schemas

The CybOX XML schema has been under development since 2012 by an ad hoc committee of interested stakeholders, led by MITRE. The binding rules discussed in the previous section are unlikely to produce identical XML schemas to those developed manually. However, the semantics must be the same. Some of the choices made when developing the original XML schema or the UML model might seem like exceptions to the binding rules. Additionally, the rules are not fully specified so they allow for arbitrary choices in the implementation.

In an effort to create an XML implementation that takes these idiosyncrasies into account, we discuss them in this section.

## UML Package to XML Namespace Name Mapping

The names of the UML Package, the XML Namespace and XML Namespace prefix are a variation of each other. UML package names and XML namespace prefixes often are the same. All XML namespaces begin with http://docs.oasis-open.org/cti/ns/cybox/and have the “-2” suffix.

For object packages, the general rule is as follows:

The name of the package is capitalized with the use of underscores. The name of the XML namespace is camel-case, removing the underscores, adding the prefix “/object/”. The name of the XML namespace prefix is camel-case but the text “Object” is shortened to “Obj”.

Package Name: File\_Object

XML Namespace: http://docs.oasis-open.org/cti/ns/cybox/objects/file-2

XML Namespace Prefix: FileObj

Non-object package names have the following mapping.

Table ‑. UML Package Names Mapping

|  |  |
| --- | --- |
| UML Package/ XML Namespace Prefix | XML Namespace |
| cyboxCommon | common |
| Cybox | core |
| cyboxVocabs | vocabularies |

The extensions package is discussed in Section **3.4**.

## UML Abstract Classes

The following XSD complex types should be declared as abstract.

From the cyboxCommon package:

* ToolSpecificDataType
* ObjectPropertiesType
* BaseObjectPropertyType
* DateTimeObjectPropertyRestrictionType
* TimeObjectPropertyRestrictionType

From the cybox package:

* DomainSpecificObjectPropertiesType
* DefinedEffectType

From the object packages:

* FileObj:FileAttributeType
* FileObj:FilePermissionsType
* ProcessObj:ProcessStatusType
* UserAccountObj:PrivilegeType
* UserAccountObj:GroupType
* VolumeObj:VolumeOptionsType

## Using XSD Data Types

**Table 3‑2** defines a mapping between a UML data type and its equivalent XSD Data Type [W3-DT] defined in the xs namespace. Because the corresponding XSD data type has its own XSD simple type definition in the xs namespace, it is not necessary to define them in the CybOX XML implementation.

Table ‑. Common basic data types

|  |  |  |  |
| --- | --- | --- | --- |
| **UML Data Type** | **Derived from BasicString** | **Definition** | **XSD Data Type** |
| BasicString | n/a | The BasicString data type is a sequence of characters. Currently, characters are defined using the UTF-8 character encoding. The number of characters allowed is finite, but unbounded. | xs:string |
| Boolean | No | The Boolean data type is defined with two possible literals: ‘*true*’ and ‘*false*’. | xs:boolean |
| Decimal | No | The Decimal data type is a sequence of decimal digits, with perhaps an intervening decimal point, “.”. The number of digits on either side of the decimal point is finite, but unbounded. Often used to express currency amounts. | xs:decimal |
| Integer | No | The Integer data type is a sequence of decimal digits, with perhaps a leading minus sign “-“. The number of decimal digits allowed is finite, but unbounded. | xs:integer |
| NonNegativeInteger | No | The NonNegativeInteger data type is a restriction on the Integer data type such that the leading minus sign is not allowed. | xs:nonNegativeInteger |
| PositiveInteger | No | The PositiveInteger data type is a restriction on the NonNegativeInteger data type that disallows zero (0). | xs:positiveInteger |
| DateTime | Yes | The DateTime data type is a restriction on the BasicString data type such that it adheres to the standard defined in http://www.iso.org/iso/home/standards/iso8601.htm | xs:dateTime |
| HexBinary | Yes | The HexBinary data type is a restriction on the BasicString data type such that it adheres to the regular expression [0-9A-Fa-f]\*. The number of characters allowed is finite but unbounded. The number of digits must be even in length. | xs:hexBinary |
| LanguageCode | Yes | The LanguageCode data type is a restriction on the BasicString data type, such that it adheres to the standard defined in [[RFC5646](http://tools.ietf.org/html/rfc5646)]. | xs:language |
| QualifiedName | Yes | The QualifiedName data type is a restriction on the BasicString data type such that it adheres to the requirements specified in [[W3Name](http://www.w3.org/TR/REC-xml-names/)]. | xs:Qname |
| NoEmbeddedQuoteString | Yes | The NoEmbeddedQuoteString data type is a restriction on the BasicString data type such that it does not include any double quote characters. This data type captures properties that were attributes in the XML model. | xs:string, but only used for XSD attributes, therefore it is implied |
| URI | Yes | The URI data type is a restriction on the BasicString data type such that it adheres to the standard defined at [http://tools.ietf.org/html/rfc 3986](http://tools.ietf.org/html/rfc%203986). | xs:anyURI |

Notice, that the definitions of the UML data types above are as similar as possible to the XSD data type definitions.

## Extensions and Externally Defined Data Models

Many data models are already externally defined, so there is no need to model them directly in CybOX. Other domains do not have an established data model, but their definition is outside the scope of the CybOX data model. In both cases, we would like to support their inclusion into CybOX instances.

The use of externally defined data models is supported by the CybOX Default Extension data model. Extensions either provide a “connection” to these established data models, or define simple extension points when no data model exists. These extension points usually, but not always, are realized by a simple UML class which can accommodate the domain once an established model is created or standardized. For more details, see [*CybOXTM Version 2.1.1 Part 04: Default Extensions*](#related_work).

The current established externally defined data models are:

Table ‑. Externally defined data models.

|  |  |
| --- | --- |
| Externally Defined Data Model | Domain |
| cpe | Platform |
| oasis\_ciq | Location |

All of these externally defined data models use XSD as the normative definition, so it is not necessary to include any additional XSD in the CybOX XML binding.[[2]](#endnote-2) The XSD for these externally defined data models can be found in the extensions directory of the current XSD model.

However, the extensions UML package must be implemented in XML. The extensions package is actually composed of two sub-packages. Both of these sub-packages should be implemented using its own XML schema, in a separate file. If the sub-package references an existing externally defined data model, it should use the xs:import construct to include the XML schema.

Table ‑: UML Extension Package Names Mapping

|  |  |
| --- | --- |
| UML Package/ XML Namespace Prefix | XML Namespace |
| stix-ciqaddress | ../location/ciq-address-3.0 |
| cpe23 | ../platform/cpe-2.3 |

All XML namespaces begin with http://docs.oasis-open.org/cti/ns/cybox/and have the “-2” suffix.

## XSD attributeGroups



Figure ‑. Using XSD attributeGroups

The BaseObjectPropertyType data type is a special case, and is represented in the CybOX XML schemas using the xs:attributeGroup construct. Since it is a special case, the expected XSD syntax is presented directly. Notice that neither of the XSD attribute groups is an extension of any other concept, however, its default value for the datatype attribute is string.

<xs:complexType name="BaseObjectPropertyType" abstract="true">

<xs:simpleContent>

<xs:extension base="xs:anySimpleType">

<xs:attributeGroup ref="cyboxCommon:BaseObjectPropertyGroup"/>

<xs:attributeGroup ref="cyboxCommon:PatternFieldGroup"/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

<xs:attributeGroup name="BaseObjectPropertyGroup">

<xs:attribute name="id" type="xs:QName"/>

<xs:attribute name="idref" type="xs:QName"/>

<xs:attribute name="datatype" type="cyboxCommon:DatatypeEnum"

default="string"/>

<xs:attribute name="appears\_random" type="xs:boolean"/>

<xs:attribute name="is\_obfuscated" type="xs:boolean"/>

<xs:attribute name="obfuscation\_algorithm\_ref" type="xs:anyURI"/>

<xs:attribute name="is\_defanged" type="xs:boolean"/>

<xs:attribute name="defanging\_algorithm\_ref" type="xs:anyURI"/>

<xs:attribute name="refanging\_transform\_type" type="xs:string"/>

<xs:attribute name="refanging\_transform" type="xs:string"/>

<xs:attribute name="observed\_encoding" type="xs:string"/>

</xs:attributeGroup>

<xs:attributeGroup name="PatternFieldGroup"/>

<xs:attribute name="condition" type="cyboxCommon:ConditionTypeEnum"/>

<xs:attribute name="is\_case\_sensitive" type="xs:boolean" default="true"/>

<xs:attribute name="apply\_condition"

type="cyboxCommon:ConditionApplicationEnum" default="ANY"/>

<xs:attribute name="delimiter" type="xs:string" default="##comma##"/>

<xs:attribute name="bit\_mask" type="xs:hexBinary"/>

<xs:attribute name="pattern\_type" type="cyboxCommon:PatternTypeEnum"/>

<xs:attribute name="regex\_syntax" type="xs:string"/>

<xs:attribute name="has\_changed" type="xs:boolean"/>

<xs:attribute name="trend" type="xs:boolean"/>

</xs:attributeGroup>

Another special case is the UML class AccessPermissionsGroup, which is also represented in the CybOX XML schemas using the xs:attributeGroup construct. In this case, the UML class which is a generalization of WindowNetworkShareObjectType, is implemented using the XSD attributeGroup construct.

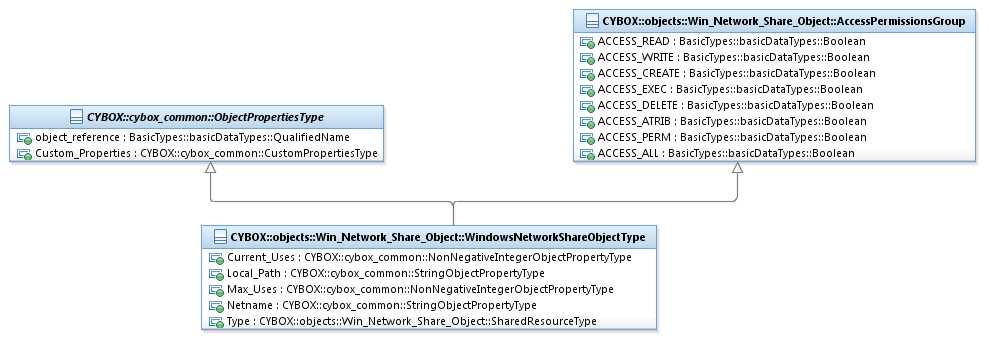


Figure ‑. UML Model for AccessPermissionsGroup

<xs:complexType name="WindowsNetworkShareObjectType" mixed="false">

<xs:complexContent>

<xs:extension base="cyboxCommon:ObjectPropertiesType">

<xs:sequence>

<xs:element name="Current\_Uses"

type="cyboxCommon:NonNegativeIntegerObjectPropertyType"

minOccurs="0"/>

<xs:element name="Local\_Path"

type="cyboxCommon:StringObjectPropertyType" minOccurs="0"/>

<xs:element name="Max\_Uses"

type="cyboxCommon:NonNegativeIntegerObjectPropertyType"

minOccurs="0"/>

<xs:element name="Netname"

type="cyboxCommon:StringObjectPropertyType" minOccurs="0"/>

<xs:element name="Type"

type="WinNetworkShareObj:SharedResourceType"

minOccurs="0"/>

</xs:sequence>

<xs:attributeGroup ref="WinNetworkShareObj:AccessPermissionsGroup"/>

</xs:extension>

</xs:complexContent>

</xs:complexType>

<xs:attributeGroup name="AccessPermissionsGroup">

<xs:attribute name="ACCESS\_READ" type="xs:boolean"/>

<xs:attribute name="ACCESS\_WRITE" type="xs:boolean"/>

<xs:attribute name="ACCESS\_CREATE" type="xs:boolean"/>

<xs:attribute name="ACCESS\_EXEC" type="xs:boolean"/>

<xs:attribute name="ACCESS\_DELETE" type="xs:boolean"/>

<xs:attribute name="ACCESS\_ATRIB" type="xs:boolean"/>

<xs:attribute name="ACCESS\_PERM" type="xs:boolean"/>

<xs:attribute name="ACCESS\_ALL" type="xs:boolean"/>

</xs:attributeGroup>

# Conformance

The last numbered section in the specification must be the Conformance section. Conformance Statements/Clauses go here.

See “Guidelines to Writing Conformance Clauses”:   
<http://docs.oasis-open.org/templates/TCHandbook/ConformanceGuidelines.html>.

1. Acknowledgments

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

Participants:

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

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

1. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Editor** | **Changes Made** |
| [Rev number] | [Rev Date] | [Modified By] | [Summary of Changes] |

1. Aggregation is not explicitly specified in the STIX UML model, so there are no binding rules associated with it. The STIX data model has a concept of embedding vs referencing, which is orthogonal to aggregation. [↑](#endnote-ref-1)
2. In fact, the STIX UML specification makes no attempt to include usable UML for these externally defined data models. The STIX UML specification does include some UML for these data models, but they were produced simply by automatically converting the XSD schemas to UML. [↑](#endnote-ref-2)