STIXTM Version 1.2.1 XML Binding Specification Version 1.0

Working Draft 01

21 January 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:

Sean Barnum ([sbarnum@mitre.org](mailto:sbarnum@mitre.org)), [MITRE Corporation](http://www.mitre.org/)

Aharon Chernin ([achernin@soltra.com](mailto:achernin@soltra.com)), [Soltra](http://www.soltra.com/)

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:

Related work:

This specification is related to:

* *STIXTM Version 1.2.1*. [URL].

Declared XML namespaces:

* list namespaces declared within this specification

Abstract:

This specification describes XML bindings for STIX Version 1.2.1.

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/stix-1.2.1-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](#_Toc450824648)

[1.1 Document Conventions 4](#_Toc450824649)

[1.1.1 Fonts 4](#_Toc450824650)

[1.1.2 XML Namespaces 4](#_Toc450824651)

[1.1.3 UML Diagram Icons and Arrow Types 5](#_Toc450824652)

[1.1.4 Color Coding 5](#_Toc450824653)

[1.1.5 XSD Examples 5](#_Toc450824654)

[2 Binding Rules 6](#_Toc450824655)

[2.1 UML Packages 6](#_Toc450824656)

[2.2 UML Classes 6](#_Toc450824657)

[2.3 UML Attributes and Associations 8](#_Toc450824658)

[2.4 UML Data Types 9](#_Toc450824659)

[2.5 UML Enumerations 10](#_Toc450824660)

[2.6 UML Interfaces 11](#_Toc450824661)

[2.7 Controlled Vocabularies 12](#_Toc450824662)

[3 Relationships to the STIX 1.2.1 XML Schemas 14](#_Toc450824663)

[3.1 UML Package to XML Namespace Name Mapping 14](#_Toc450824664)

[3.2 UML Classes 14](#_Toc450824665)

[3.2.1 Abstract XSD Complex Types 14](#_Toc450824666)

[3.3 UML Data Types 15](#_Toc450824667)

[3.3.1 Using XSD Data Types 15](#_Toc450824668)

[3.3.2 UML Data Types Explicitly Defined in the XML Implementation 16](#_Toc450824669)

[3.4 UML Interfaces 18](#_Toc450824670)

[3.5 Controlled Vocabularies 18](#_Toc450824671)

[3.6 Extensions and Externally Defined Data Models 20](#_Toc450824672)

[4 Conformance 22](#_Toc450824673)

[Appendix A. Acknowledgments 23](#_Toc450824674)

[Appendix B. Revision History 24](#_Toc450824675)

# Introduction

[All text is normative unless otherwise labeled]

The Structured Threat Information Expression (STIXTM) specification consists of a formal UML model and a set of textual specification documents that explain the UML model. Specification documents have been written for each of the individual data models that compose the full STIX UML model, which in addition to the nine top-level component data models (Observable[[1]](#endnote-1), Indicator, Incident, TTP, ExploitTarget, CourseOfAction, Campaign, ThreatActor, and Report), includes a core data model, a common data model, a default extension data model, a data marking data model, and a set of default controlled vocabularies.

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 mentioned above.

In Section 2, we define a binding rule for each concept in UML that was used in the STIX specification. Because the STIX 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 STIX 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 STIX. The XML schemas were developed previously to the UML model and are officially defined as the normative XML schema definition for STIX 1.2.1.  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 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 STIX high level concepts.

Examples: Indicator, Course of Action, Threat Actor

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

Examples: xs:complexType, xs:string, incident:LossEstimationType:amount

### XML Namespaces

Each STIX data model is captured in a different XML schema, related to the UML packages which together compose the full STIX UML model. The STIX 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. |

### 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 collection of specification documents via exemplars are illustrated in **Figure 1‑1**.



Figure ‑. Data model color coding

### XSD Examples

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

# 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 names used. The concept of a namespace in XSD is similar. Because STIX contains nine top-level concepts, it makes sense that each of them would be represented using an individual UML package. Other UML packages correspond to the overarching data models in STIX, such as Core and Common. The use of multiple packages allows the STIX data model to be modular: all of the STIX components are defined in separate packages rather than in one large package to limit interdependence between STIX 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/stix*/<namespace-name>*-1
* The prefix name must be declared. The package name and prefix name are the same.
  + xmlns:<*package-name*>="http://docs.oasis-open.org/cti/ns/stix/<*namepsace-name*>-1”
* Other XML schemas of the STIX model must be imported as needed, using the xs:import syntax. The namespace prefix of any imported schema must also be specified.

The CybOX data model is used extensively in the STIX data model. For that reason, most schemas should import the necessary CybOX Core and Common XML schemas, and their corresponding namespace prefixes. This document assumes that the XML schemas for CybOX exist.

## UML Classes

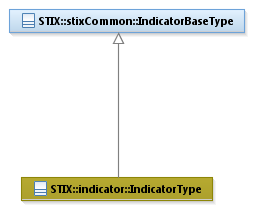
A *class* in UML corresponds to xs:complexType in XSD. In UML, classes are related to each other using *generalization*, which indicates that one is derived from the other. In XSD, generalization corresponds to an *extension* of the type.

**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 syntax.
* Complex types that are extensions of other complex types must use the xs:complexContent syntax, as shown in the example below.
* Certain XSD complex types should be declared as abstract. See Section 3.2.1 for guidance.
* All xs:element syntax must be enclosed within the xs:sequence syntax.

**Exception 1:** The xs:choice syntax is used twice within the STIX 1.2.1 XML schemas. In both cases, they will contain xs:element tags. See Section **3.4** for more details.

Here is a generalization relationship from the UML model:



Here is the equivalent definition from the XSD:

<xs:complexType name="IndicatorType">

<xs:complexContent>

<xs:extension base="stixCommon:IndicatorBaseType">

<!-- snip! -->

</xs:complexContent>

</xs:complexType>

XSD simpleTypes 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[[2]](#endnote-2).

**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 3‑1**.
* If the type of the UML attribute is stixCommon:VocabularyStringType, then the type of the corresponding XSD element must be stixCommon:ControlledVocabularyStringType.
* The type of an XSD attribute can either be a XSD simple type defined elsewhere (usually corresponding to a UML data type defined in the basicDataTypes package), or in-place via use of the xs:restriction syntax. See **Table 3‑2** for guidance and Section **2.4** for an example.
* 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:

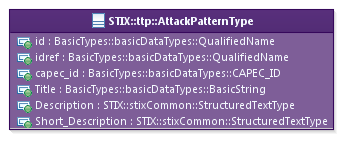


Figure ‑. UML AttackPatternType class

Here is the equivalent definition from the XSD (with some details elided):

<xs:complexType name="AttackPatternType">

<xs:sequence>

<xs:element name="Title" type="xs:string" minOccurs="0"/>

<xs:element name="Description" type="stixCommon:StructuredTextType" minOccurs="0"/>

<xs:element name="Short\_Description" type="stixCommon:StructuredTextType" minOccurs="0"/>

</xs:sequence>

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

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

<xs:attribute name="capec\_id">

<!-- snip! -->

</xs:attribute>

</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 UML data types are used in various data models in addition to STIX, we introduced a separate UML package, basicDataTypes, to hold most of the UML data types.

**Binding Rule 4**: 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 correspond to strings which have semantics associated with them, usually the strings are restricted to a certain pattern. The pattern is defined via a regular expression, and/or more formally, in a standardization document. These are implemented as XSD simple types, that make use of the xs:restriction and xs:pattern syntax.
* For XSD simple types that will only be used once, there is no need to name the simple type, and it can be defined “in-place” (see capec\_id below). Other times, when a simple type is used to define a XSD element it uses a named simple type defined elsewhere. See **Table 3‑2** for guidance.

See **Figure 2‑1** (above) for examples of UML attributes that are of data types from BasicDataTypes. In XSD, they would be expressed as XSD data types or simple types.

<xs:complexType name="AttackPatternType">

<!-- SNIP!! -->

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

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

<xs:attribute name="capec\_id">

<xs:simpleType>

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

<xs:pattern value="CAPEC-\d+"/>

</xs:restriction>

</xs:simpleType>

</xs:attribute>

</xs:complexType>

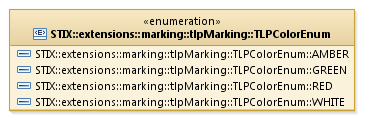
## UML Enumerations

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., colors in a traffic light).

**Binding Rule 5:** The XML binding for a UML enumeration is an XSD simple type using the xs:enumeration syntax.

* *This binding rule only applies to UML enumerations not used to model STIX 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:



This would be represented in XSD as:

<xs:simpleType name="TLPColorEnum">

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

<xs:enumeration value="RED"/>

<xs:enumeration value="AMBER"/>

<xs:enumeration value="GREEN"/>

<xs:enumeration value="WHITE"/>

</xs:restriction>

</xs:simpleType>

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

## UML Interfaces

UML interfaces are rarely used in the UML data model for STIX. However, they are useful to describe the XSD choice construct. This is the only use made of UML interfaces.

The following UML diagram describes the use of a UML interface to model the semantics of XSD choice.

**Binding Rule 6:** The XML binding for a UML interface corresponds to an XSD choice.

* All of the realizations of the UML interface must correspond to an element in an XSD choice of the realized XSD type. This XSD choice becomes part of the XSD complex type that is bound to the overarching UML class.
* The names of the choice elements are based on the complex type names, by adding underscores and omitting “Type”.



The XSD syntax that this corresponds to is:

<xs:choice>

<xs:element name="Observable" type="cybox:ObservableType" minOccurs="0"/>

<xs:element name="Composite\_Indicator\_Expression"

type="indicator:CompositeIndicatorExpressionType"

minOccurs="0"/>

</xs:choice>

<xs:complexType name="CompositeIndicatorExpressionType">

<xs:sequence>

<xs:element ref="indicator:Indicator" minOccurs="0"

maxOccurs="unbounded"/>

</xs:sequence>

<xs:attribute name="operator" type="indicator:OperatorTypeEnum"

use="required"/>

</xs:complexType>

In general, there isn’t a correspondence between UML interfaces to any XSD construct, because XSD is used to model data, not functionality (i.e. methods).

## Controlled Vocabularies

The binding rule for the UML model of STIX 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.

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 STIX community.
2. Formally define and leverage a custom vocabulary. Producers and consumers agree upon a common vocabulary that they use in the sharing of STIX 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.

**Binding Rule 7a:** Introduce an XSD complex type stixCommon:ControlledVocabularyStringType, to encapsulate all of the use cases supported by UML data types stixCommon:VocabularyStringType, stixCommon:UnenforcedVocabularyStringType and stixCommon: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 stixVocabs:

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

Here is the XSD for defining controlled vocabularies, using HighMediumLowVocab 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 stixVocabs:ControlledVocabularyStringType as its type. This allows contributors of STIX content to be able to specify values for a controlled vocabulary that satisfy all use cases from the above specification.

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

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

<xs:enumeration value="High"/>

<xs:enumeration value="Medium"/>

<xs:enumeration value="Low"/>

<xs:enumeration value="None"/>

<xs:enumeration value="Unknown"/>

</xs:restriction>

</xs:simpleType>

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

<xs:simpleContent>

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

<xs:simpleType>

<xs:union memberTypes="stixVocabs:HighMediumLowEnum-1.0"/>

</xs:simpleType>

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

fixed="STIX Default High/Medium/Low Vocabulary"/>

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

use="optional"

fixed "http://docs.oasis-open.org/cti/stix-1.2.1-xml-

binding/v1.0/csd01/schemas/vocabularies.xsd#HighMediumLowVocab-1.0"/>

</xs:restriction>

</xs:simpleContent>

</xs:complexType>

# Relationships to the STIX 1.2.1 XML Schemas

The STIX 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 the same except for the following table. UML package names and XML namespace prefixes often are the same.

Table ‑: UML Package Names Mapping

|  |  |
| --- | --- |
| UML Package/ XML Namespace Prefix | XML Namespace |
| coa | course-of-action |
| et | exploit-target |
| stix | core |
| stixCommon | common |
| stixVocabs | vocabularies |
| ta | threat-actor |
| marking | data-marking |

All XML namespaces begin with http://docs.oasis-open.org/cti/ns/stix/and have the “-1” suffix.

The extensions package contains many sub-packages – each with their own mappings. This is discussed in Section **3.6**.

## UML Classes

### Abstract XSD Complex Types

The following XSD complex types should be declared as abstract.

From the stixCommon package:

* ActivityType
* GenericRelationshipListType
* GenericRelationshipType
* AddressAbstractType

From the coa package:

* StructuredCOAType

From the incident package:

* ExternalImpactAssessmentModelType

From the indicator package:

* TestMechanismType

## UML Data Types

### 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 STIX 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.

**Exception 2:** The only UML attribute that is declared to be of data type Decimal is incident:LossEstimationType:amount. In STIX 1.2.1, the corresponding XSD attribute has no xsd:type.

### UML Data Types Explicitly Defined in the XML Implementation

The UML data types in **Table 3‑3** correspond to strings that have semantics associated with them. Because of this, they are restricted to a certain pattern, defined via a regular expression.

There are no predefined XSD data types that correspond to these UML data types, so they must be explicitly defined in the XML implementation using XSD simple types. If the data type is only used once, if it sufficient to define the XSD simple type “in place,” with no need to name the XSD simple type.

Table ‑: List of explicitly defined XSD data types

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Type** | **Definition** | **Named?** |  |
| CAPEC\_ID | The CAPEC\_ID data type is a restriction on the BasicString data type, such that it adheres to the regular expression “CAPEC-\d+”. The CAPEC\_ID values should correspond to those defined at <http://capec.mitre.org>. | No | xs:string, with xs:pattern restriction |
| CCE\_ID | The CCE\_ID data type is a restriction on the BasicString data type such that it adheres to the regular expression “CCE-\d+\d”. The CCE\_ID values should correspond to those defined at <http://cce.mitre.org>. | No | xs:string, with xs:pattern restriction |
| CVE\_ID | The CVE\_ID data type is a restriction on the BasicString data type such that it adheres to the regular expression “CVE-\d\d\d\d+\d+”. The CVE\_ID values should correspond to those defined at <http://cve.mitre.org>. | No | xs:string, with xs:pattern restriction |
| CWE\_ID | The CWE\_ID data type is a restriction on the BasicString data type such that it adheres to the regular expression “CWE-\d+”. The CWE\_ID values should correspond to those defined at <http://cwe.mitre.org>. | No | xs:string, with xs:pattern restriction |
| et:CVSSScoreType | The CVSSScoreType data type is a restriction on the BasicString data type, such that it adheres to the regular expression "((10)|[0-9])\.[0-9]". | Yes | xs:string, with xs:pattern restriction |
| et:CVSSBaseVectorType | The CVSSBaseVectorType data type is a restriction on the BasicString data type, such that it adheres to the regular expression AV:[LAN]/AC:[HML]/Au:[MSN]/C:[NPC]/I:[NPC]/A:[NPC]". | Yes | xs:string, with xs:pattern restriction |
| et:CVSSTemporalVectorType | The CVSSBaseVectorType data type is a restriction on the BasicString data type, such that it adheres to the regular expression “E:([UFH]|(POC)|(ND))/RL:([WU]|(OF)|(TF)|(ND))/RC:([C]|(UC)|(UR)|(ND))". | Yes | xs:string, with xs:pattern restriction |
| et:CVSSEnvironmentalVectorType | The CVSSBaseVectorType data type is a restriction on the BasicString data type, such that it adheres to the regular expression “CDP:([NLH]|(LM)|(MH)|(ND))/TD:([NLMH]|(ND))/CR:([LMH]|(ND))/IR:([LMH]|(ND))/AR:([LMH]|(ND))". | Yes | xs:string, with xs:pattern restriction |

**Exception 3**: For consistency with current XSD schemas, the UML data types listed in Table 3‑2 that have a “Yes” in the “Named?” column should be defined using separately named XSD simple types..

## UML Interfaces

UML interfaces are used to express the concept of Indicator Expressions. Indicator Expressions correspond to common Boolean expressions found in many languages/data models – i.e., basic relational expressions (Observables) or composite of expressions with a Boolean operation (AND/OR).[[3]](#endnote-3)

Note that this UML model is syntactically different than the XSD model in several ways:

* Any XSD complex type that uses this XSD choice construct would have two elements, which by definition of XSD choice are mutually exclusive – i.e., only one is valid at any one time.
* In the UML model, there is only one UML attribute, IndicatorExpression, to model this attribute, and it is of type PatternExpression, which uses UML interfaces to represent mutual exclusivity.
* PatternExpression is not part of the STIX package because it is realized by a UML class defined outside of the model (i.e., cybox:Observables).

To implement the two UML interfaces:

In the indicator XSD, insert the xs:choice syntax after the xs:element with the name Valid\_Time\_Position.

In the incident XSD, enclose the two xs:element tags within xs:choice tags.

## Controlled Vocabularies

To specify a default controlled vocabulary, an XSD complex type and an XSD simple type are used in conjunction. One of the important aspects of using a particular controlled vocabulary is that the value used must be one of the enumerated values for the content as a whole to be valid. We do this through use of the xsi:type dynamic extension mechanism. However, in XSD, an enumeration can only be defined as a restriction of the XSD simple type xs:string. Because an XSD simple type cannot extend an XSD complex type, it was necessary to introduce an XSD complex type for each default controlled vocabulary in the data model. This XSD complex type encapsulates the vocabulary enumeration simple type, and therefore can be used as an extension of ControlledVocabularyStringType.

If a XSD complex type has a XSD element that uses a vocabulary it should use stixCommon:ControlledVocabularyStringType as its type. Usually, a default controlled vocabulary is suggested. This allows contributors of STIX content to be able to specify values for a controlled vocabulary that satisfy all of the use cases specified in Section **2.7**. Below we give examples of XML instances, which covers those use cases, that are valid based on XSD derived using binding rule 7.

An example of use case 1 is:

<incident:Confidence>

<stixCommon:Value xsi:type="stixVocabs:HighMediumLowVocab-1.0">High</stixCommon:Value>

</incident:Confidence>

Assuming example:OurCustomVocab-1.0 has been defined as a new controlled vocabulary that a group of consumers and producers want to share. Even though it is not one of the default controlled vocabularies provided in the STIX data model, its definition can be imported by interested users, and the use of appropriate values will be validated. This is an example of use case 2:

<incident:Confidence>

<stixCommon:Value xsi:type="example:OurCustomVocab-1.0">Vetted</stixCommon:Value>

</incident:Confidence>

In this example of use case 3, a controlled vocabulary is defined external to the STIX data model. A link to the definition is provided as is the name of the vocabulary. However, these attributes are for documentation only. The STIX data model does not enforce verification that the values used to be members of the externally defined vocabulary.

<incident:Confidence>

<stixCommon:Value

vocab\_reference="http://myvocabname.example.com/DifferentRatingScale"

vocab\_name="DifferentRatingScale ">Unreliable</stixCommon:Value>

</incident:Confidence>

In this example of use case 4, the value for the type of a Threat Actor can be from the controlled vocabulary stixVocabs:HighMediumLowVocab-1.0, but in this case, the desired value of “Very\_High” was not a member of that controlled vocabulary. Use Case 4 allows for the use of other values when necessary.

<incident:Confidence>

<stixCommon:Value>Very\_High</stixCommon:Value>

</incident:Confidence>

When using UML to model the four use cases of controlled vocabularies, we needed to make accommodations for limitations in UML modeling. As we stated in the introduction, we tried to make the UML data model adhere as closely as possible to the XSD implementation already established. However, it proved impossible and also undesirable to model the use of a separate UML class and UML enumeration. We also wanted to avoid the introduction of an attribute for the actual vocabulary value. Additionally, we felt it was “cleaner” to model use cases 3 and 4 separately. The UML model, again using the HighMediumLow controlled vocabulary as an exemplar, is in **Figure 3‑1**.

Because a UML enumeration and a UML data type cannot both be a specialization of a UML class, we do not use UML classes to model vocabularies. To avoid introducing a new attribute, all vocabularies’ use cases are supported by being specializations of the UML data type basicDataTypes:BasicString. UnenforcedVocabularyStringType was introduced to support use case 3. Because UML data types can have attributes, there is no need to introduce a UML class to specify attributes for vocab\_name and vocab\_reference. Additionally, UML enumerations can be specializations of UML data types, and this matches nicely with our concept of controlled vocabularies – a restricted set of legal strings.

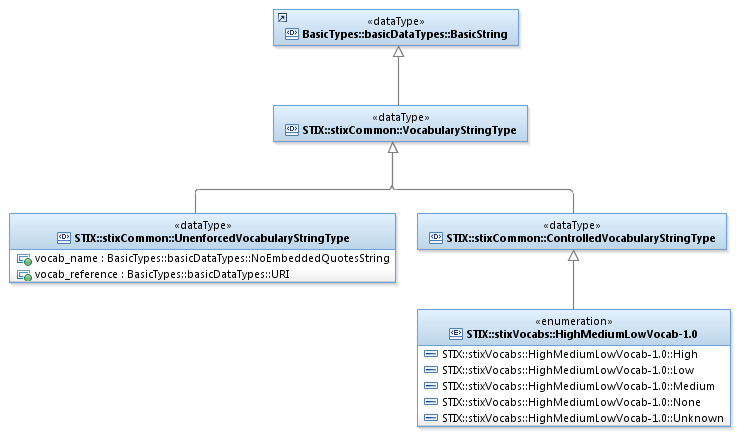


Figure ‑: Modelling controlled vocabularies in UML

## Extensions and Externally Defined Data Models

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

The use of externally defined data models is supported by the STIX 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 *STIXTM Version 1.2.1 Part 12: Default Extensions*.

The current established externally defined data models are:

Table ‑. Externally defined data models.

|  |  |
| --- | --- |
| Externally Defined Data Model | Domain |
| capec | Attack Patterns |
| cvrf | Vulnerabilities |
| maec | Malware |
| oasis\_ciq | Identity and Location |
| open\_ioc | Indicators |
| oval | Assessment |

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 STIX XML binding.[[4]](#endnote-4) 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 eight sub-packages. Each 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 syntax to include the XML schema.

The names of the UML Package, the XML Namespace and XML Namespace prefix are the same except in for the following:

Table ‑: UML Extension Package Names Mapping

|  |  |
| --- | --- |
| UML Package/ XML Namespace Prefix | XML Namespace |
| stix-capec | ../extensions/capec-2.7 |
| stix-ciqaddress | ../extensions/ciq-address-3.0 |
| stix-ciqidentity | ../extensions/ciq-3.0-identity |
| stix-cvrf | ../extensions/cvrf |
| stix-maec | ../extensions/Maec-4.1 |
| simpleMarking | ../extensions/data-marking/simple |
| TOUMarking | ../extensions/data-marking/terms-of-use |
| tlpMarking | ../extensions/data-marking/tlp |
| genericStructuredCOA | ../extensions/structure-coa/generic |
| genericTM | ../extensions/test-mechanism/generic |
| stix-openioc | ../extensions/test-mechanism/openioc-2010 |
| stix-oval | ../extensions/test-mechanism/oval-5.10 |
| snortTM | ../extensions/test-mechanism/snort |
| yaraTM | ../extensions/test-mechanism/yara |

All XML namespaces begin with http://docs.oasis-open.org/cti/ns/stix/and have the “-1” suffix.

The use of the binding rules in Section **2** is sufficient for implementing the extensions package except for these cases:

* identity package. The class STIXCIQIdentity3.0Type is used to restrict the ciq:PartyType to a subset of properties available. For a list, see *STIX Version 1.2.1 Part 12: Default Extensions*.
* test\_mechanism package. Many of the UML attributes are of the class stixCommon:NativeFormatStringType. The type of the XSD element must be stixCommon:EncodedCDATAType. Because the syntax of native format of the test\_mechanisms may allow for illegal XML, all such data must be wrapped using the CDATA XML syntax, which is the semantics of stixCommon:EncodedCDATAType.
* structured\_coa package. Similar to test\_mechanism package.

# 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:

[Participant Name, Affiliation | Individual Member]

[Participant Name, Affiliation | Individual Member]

1. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Editor** | **Changes Made** |
| Wd01 | 21 January 2016 | Sean Barnum  Aharon Chernin  Rich Piazza | Initial transfer to OASIS template |

1. The CybOX Observable data model is actually defined in the CybOX Language, not in STIX. [↑](#endnote-ref-1)
2. 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-2)
3. The NOT operator is handled within the Observable data model. [↑](#endnote-ref-3)
4. 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-4)