



# STIX<sup>TM</sup> Version 1.2.1. Part 2: Common

## Committee Specification Draft 01 / Public Review Draft 01

06 November 2015

### Specification URIs

#### This version:

<http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part2-common/stix-v1.2.1-csprd01-part2-common.docx> (Authoritative)  
<http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part2-common/stix-v1.2.1-csprd01-part2-common.html>  
<http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part2-common/stix-v1.2.1-csprd01-part2-common.pdf>

#### Previous version:

N/A

#### Latest version:

<http://docs.oasis-open.org/cti/stix/v1.2.1/stix-v1.2.1-part2-common.docx> (Authoritative)  
<http://docs.oasis-open.org/cti/stix/v1.2.1/stix-v1.2.1-part2-common.html>  
<http://docs.oasis-open.org/cti/stix/v1.2.1/stix-v1.2.1-part2-common.pdf>

#### Technical Committee:

OASIS Cyber Threat Intelligence (CTI) TC

#### Chair:

Richard Struse ([Richard.Struse@HQ.DHS.GOV](mailto:Richard.Struse@HQ.DHS.GOV)), DHS Office of Cybersecurity and Communications (CS&C)

#### Editors:

Sean Barnum ([sbarnum@mitre.org](mailto:sbarnum@mitre.org)), MITRE Corporation  
Desiree Beck ([dbeck@mitre.org](mailto:dbeck@mitre.org)), MITRE Corporation  
Aharon Chernin ([achernin@soltra.com](mailto:achernin@soltra.com)), Soltra  
Rich Piazza ([rpiazza@mitre.org](mailto:rpiazza@mitre.org)), MITRE Corporation

#### Additional artifacts:

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

- *STIX Version 1.2.1. Part 1: Overview.* <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part1-overview/stix-v1.2.1-csprd01-part1-overview.html>
- *STIX Version 1.2.1. Part 2: Common* (this document). <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part2-common/stix-v1.2.1-csprd01-part2-common.html>
- *STIX Version 1.2.1. Part 3: Core.* <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part3-core/stix-v1.2.1-csprd01-part3-core.html>
- *STIX Version 1.2.1. Part 4: Indicator.* <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part4-indicator/stix-v1.2.1-csprd01-part4-indicator.html>
- *STIX Version 1.2.1 Part 5: TTP.* <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part5-ttp/stix-v1.2.1-csprd01-part5-ttp.html>
- *STIX Version 1.2.1. Part 6: Incident.* <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part6-incident/stix-v1.2.1-csprd01-part6-incident.html>

- *STIX Version 1.2.1. Part 7: Threat Actor.* <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part7-threat-actor/stix-v1.2.1-csprd01-part7-threat-actor.html>
- *STIX Version 1.2.1. Part 8: Campaign.* <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part8-campaign/stix-v1.2.1-csprd01-part8-campaign.html>
- *STIX Version 1.2.1. Part 9: Course of Action.* <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part9-coa/stix-v1.2.1-csprd01-part9-coa.html>
- *STIX Version 1.2.1. Part 10: Exploit Target.* <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part10-exploit-target/stix-v1.2.1-csprd01-part10-exploit-target.html>
- *STIX Version 1.2.1. Part 11: Report.* <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part11-report/stix-v1.2.1-csprd01-part11-report.html>
- *STIX Version 1.2.1. Part 12: Default Extensions.* <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part12-extensions/stix-v1.2.1-csprd01-part12-extensions.html>
- *STIX Version 1.2.1. Part 13: Data Marking.* <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part13-data-marking/stix-v1.2.1-csprd01-part13-data-marking.html>
- *STIX Version 1.2.1. Part 14: Vocabularies.* <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part14-vocabularies/stix-v1.2.1-csprd01-part14-vocabularies.html>
- *STIX Version 1.2.1. Part 15: UML Model.* <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part15-uml-model/stix-v1.2.1-csprd01-part15-uml-model.html>
- UML Model Serialization: <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/uml-model/>

#### Related work:

This specification replaces or supersedes:

- *STIX<sup>TM</sup> 1.2 Common Specification (v1.2).* [https://github.com/STIXProject/specifications/blob/version1.2/documents/pdf%20versions/STIX\\_Common\\_Draft.pdf](https://github.com/STIXProject/specifications/blob/version1.2/documents/pdf%20versions/STIX_Common_Draft.pdf)

This specification is related to:

- *Cybox<sup>TM</sup> Version 2.1.1.* Work in progress. [https://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=cti-cybox](https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=cti-cybox)
- *Cybox<sup>TM</sup> 2.1.* <https://cyboxproject.github.io/>

#### Abstract:

The Structured Threat Information Expression (STIX) framework defines nine core constructs and the relationships between them for the purposes of modeling cyber threat information and enabling cyber threat information analysis and sharing. This specification document defines the Common data model, which defines base classes that are extended or leveraged by the core components, relationship-oriented classes, content aggregation classes, vocabulary-related classes, kill chain-related classes, and other classes shared by the core constructs.

#### Status:

This document was last revised or approved by the OASIS Cyber Threat Intelligence (CTI) TC on the above date. The level of approval is also listed above. Check the “Latest version” location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at [https://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=cti#technical](https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=cti#technical).

TC members should send comments on this specification to the TC’s email list. Others should send comments to the TC’s public comment list, after subscribing to it by following the instructions at the “Send A Comment” button on the TC’s web page at <https://www.oasis-open.org/committees/cti/>.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC’s web page (<https://www.oasis-open.org/committees/cti/ipr.php>).

**Citation format:**

When referencing this specification the following citation format should be used:

**[STIX-v1.2.1-Common]**

*STIX<sup>TM</sup> Version 1.2.1. Part 2: Common.* Edited by Sean Barnum, Desiree Beck, Aharon Chernin, and Rich Piazza. 06 November 2015. OASIS Committee Specification Draft 01 / Public Review Draft 01. <http://docs.oasis-open.org/cti/stix/v1.2.1/csprd01/part2-common/stix-v1.2.1-csprd01-part2-common.html>. Latest version: <http://docs.oasis-open.org/cti/stix/v1.2.1/stix-v1.2.1-part2-common.html>.

---

# Notices

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](#) 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.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of [OASIS](#), the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <https://www.oasis-open.org/policies-guidelines/trademark> for above guidance.

Portions copyright © United States Government 2012-2015. All Rights Reserved.

STIX<sup>™</sup>, TAXII<sup>™</sup>, AND CybOX<sup>™</sup> (STANDARD OR STANDARDS) AND THEIR COMPONENT PARTS ARE PROVIDED "AS IS" WITHOUT ANY WARRANTY OF ANY KIND, EITHER EXPRESSED, IMPLIED, OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY THAT THESE STANDARDS OR ANY OF THEIR COMPONENT PARTS WILL CONFORM TO SPECIFICATIONS, ANY

IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR FREEDOM FROM INFRINGEMENT, ANY WARRANTY THAT THE STANDARDS OR THEIR COMPONENT PARTS WILL BE ERROR FREE, OR ANY WARRANTY THAT THE DOCUMENTATION, IF PROVIDED, WILL CONFORM TO THE STANDARDS OR THEIR COMPONENT PARTS. IN NO EVENT SHALL THE UNITED STATES GOVERNMENT OR ITS CONTRACTORS OR SUBCONTRACTORS BE LIABLE FOR ANY DAMAGES, INCLUDING, BUT NOT LIMITED TO, DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES, ARISING OUT OF, RESULTING FROM, OR IN ANY WAY CONNECTED WITH THESE STANDARDS OR THEIR COMPONENT PARTS OR ANY PROVIDED DOCUMENTATION, WHETHER OR NOT BASED UPON WARRANTY, CONTRACT, TORT, OR OTHERWISE, WHETHER OR NOT INJURY WAS SUSTAINED BY PERSONS OR PROPERTY OR OTHERWISE, AND WHETHER OR NOT LOSS WAS SUSTAINED FROM, OR AROSE OUT OF THE RESULTS OF, OR USE OF, THE STANDARDS, THEIR COMPONENT PARTS, AND ANY PROVIDED DOCUMENTATION. THE UNITED STATES GOVERNMENT DISCLAIMS ALL WARRANTIES AND LIABILITIES REGARDING THE STANDARDS OR THEIR COMPONENT PARTS ATTRIBUTABLE TO ANY THIRD PARTY, IF PRESENT IN THE STANDARDS OR THEIR COMPONENT PARTS AND DISTRIBUTES IT OR THEM "AS IS."

---

# Table of Contents

1	Introduction .....	8
1.1	STIX <sup>™</sup> Specification Documents .....	8
1.2	Document Conventions .....	9
1.2.1	Fonts.....	9
1.2.2	UML Package References .....	9
1.2.3	UML Diagrams.....	9
1.2.4	Property Table Notation .....	11
1.2.5	Property and Class Descriptions .....	11
1.3	Terminology .....	12
1.4	Normative References .....	12
2	Background Information .....	13
3	STIX <sup>™</sup> Common Data Model.....	14
3.1	Component Base Classes .....	14
3.1.1	CampaignBaseType Class.....	15
3.1.2	CourseOfActionBaseType Class.....	15
3.1.3	ExploitTargetBaseType Class .....	16
3.1.4	IncidentBaseType Class.....	17
3.1.5	IndicatorBaseType Class .....	18
3.1.6	ThreatActorBaseType Class .....	19
3.1.7	TTPBaseType Class .....	19
3.1.8	ReportBaseType Class .....	20
3.2	Relationship-Oriented Classes .....	21
3.2.1	GenericRelationshipType Class.....	21
3.2.2	RelatedCampaignType Class.....	22
3.2.3	RelatedCampaignReferenceType Class.....	23
3.2.4	RelatedCourseOfActionType Class .....	25
3.2.5	RelatedExploitTargetType Class.....	27
3.2.6	RelatedIdentityType Class .....	28
3.2.7	RelatedIncidentType Class .....	29
3.2.8	RelatedIndicatorType Class .....	30
3.2.9	RelatedObservableType Class.....	32
3.2.10	RelatedPackageRefType Class .....	33
3.2.11	RelatedReportType Class .....	34
3.2.12	RelatedThreatActorType Class .....	36
3.2.13	RelatedTTPType Class .....	37
3.3	Content Aggregation Classes.....	38
3.3.1	GenericRelationshipListType .....	38
3.3.2	ConfidenceAssertionChainType Class.....	39
3.3.3	ContributingSourcesType Class.....	40
3.3.4	ExploitTargetsType Class .....	41
3.3.5	NamesType Class .....	41
3.3.6	ProfilesType Class .....	42
3.3.7	ReferencesType Class .....	42

3.3.8 RelatedIdentitiesType Class .....	43
3.3.9 RelatedPackageRefsType Class .....	43
3.4 Kill Chains .....	43
3.4.1 KillChainsType Class .....	43
3.4.2 KillChainPhasesReferenceType Class .....	45
3.5 General Shared Classes .....	46
3.5.1 ActivityType Class .....	46
3.5.2 AddressAbstractType Class .....	47
3.5.3 ConfidenceType Class .....	48
3.5.4 IdentityType Class .....	49
3.5.5 InformationSourceType Class .....	50
3.5.6 StatementType Class .....	51
3.5.7 ToolInformationType Class .....	53
3.6 General Data Types .....	54
3.6.1 DateTimeWithPrecisionType Data Type .....	54
3.6.2 NativeFormatStringType Data Type .....	55
3.6.3 StructuredTextType Data Type .....	55
3.7 Vocabulary Data Types .....	56
3.7.1 VocabularyStringType Data Type .....	58
3.7.2 UnenforcedVocabularyStringType Data Type .....	58
3.7.3 ControlledVocabularyStringType Data Type .....	58
3.8 Enumerations .....	59
3.8.1 DateTimePrecisionEnum Enumeration .....	59
3.8.2 RelationshipScopeEnum .....	59
4 Conformance .....	61
Appendix A. Acknowledgments .....	62
Appendix B. Revision History .....	64

---

# 1 Introduction

[All text is normative unless otherwise labeled]

The Structured Threat Information Expression (STIX<sup>™</sup>) framework defines nine top-level component data models: Observable<sup>1</sup>, Indicator, Incident, TTP, ExploitTarget, CourseOfAction, Campaign, ThreatActor, and Report. In addition, it defines a common data model that defines base classes that are extended or leveraged by the core components, relationship-oriented classes, content aggregation classes, kill chain-related classes, and other classes shared by the core constructs. This document serves as the specification for the STIX Common data model.

The STIX Common data model defines object classes that are shared across the various STIX data models. For clarity in the document, the classes are divided into five types: component base classes (Section 3.1), relationship-oriented classes (Section 3.2), content aggregation classes (Section 3.3), kill chain-related classes (Section 3.4), and general shared classes (Section 3.5). We also list general data types (Section 3.6), vocabulary-related data types (Section 3.7), and enumerations separately (Section 3.8).

In Section 1.1 we discuss additional specification documents, in Section 1.2 we provide document conventions, and in Section 1.3 we provide terminology. References are given in Section 1.4. In Section 2, we give background information to help the reader better understand the specification details that are provided later in the document. We present the Common data model specification details in Section 3 and conformance information in Section 4.

## 1.1 STIX<sup>™</sup> Specification Documents

The STIX 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 data models that compose the full STIX UML model.

The *STIX Version 1.2.1 Part 1: Overview* document provides a comprehensive overview of the full set of STIX data models, which in addition to the nine top-level component data models mentioned in the Introduction, includes a core data model, a common data model, a cross-cutting data marking data model, various extension data models and a set of default controlled vocabularies. *STIX Version 1.2.1 Part 1: Overview* also summarizes the relationship of STIX to other languages and outlines general STIX data model conventions.

**Figure 1-1** illustrates the [set of specification documents](#) that are available. The color black is used to indicate the specification overview document, altered shading differentiates the overarching Core data model from the supporting data models (vocabularies, data marking, and default extensions), and the color white indicates the component data models. The solid grey color denotes the overall STIX Language UML model. This Common specification document is highlighted in its associated color (see Section 1.2.3.3). For a list of all STIX documents and related information sources, please see *STIX Version 1.2.1 Part 1: Overview*.





Figure 1-1. STIX<sup>TM</sup> Language v1.2.1 specification documents

## 1.2 Document Conventions

The following conventions are used in this document.

### 1.2.1 Fonts

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

- Capitalization is used for STIX high level concepts, which are defined in [STIX Version 1.2.1 Part 1: Overview](#).

Examples: Indicator, Course of Action, Threat Actor

- The `Courier New` font is used for writing UML objects.

Examples: `RelatedIndicatorsType`, `stixCommon:StatementType`

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

- The '*italic*' font (with single quotes) is used for noting actual, explicit values for STIX Language properties. The *italic* font (without quotes) is used for noting example values.

Example: *'PackageIntentVocab-1.0,' high, medium, low*

### 1.2.2 UML Package References

Each STIX data model is captured in a different UML package (e.g., Core package, Campaign package, etc.) where the packages together compose the full STIX UML model. To refer to a particular class of a specific package, we use the format `package_prefix:class`, where `package_prefix` corresponds to the appropriate UML package. [STIX Version 1.2.1 Part 1: Overview](#) contains a list of the packages used by the Common data model, along with the associated prefix notations, descriptions, examples.

Note that in this specification document, we do not explicitly specify the package prefix for any classes that originate from the Common data model.

### 1.2.3 UML Diagrams

This specification makes use of UML diagrams to visually depict relationships between STIX Language constructs. Note that the diagrams have been extracted directly from the full UML model for STIX; 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 data types. 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.






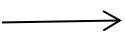

### 1.2.3.1 Class Properties

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.

### 1.2.3.2 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 1-1. UML diagram icons

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

### 1.2.3.3 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 Common specification are illustrated via exemplars in [Figure 1-2](#).

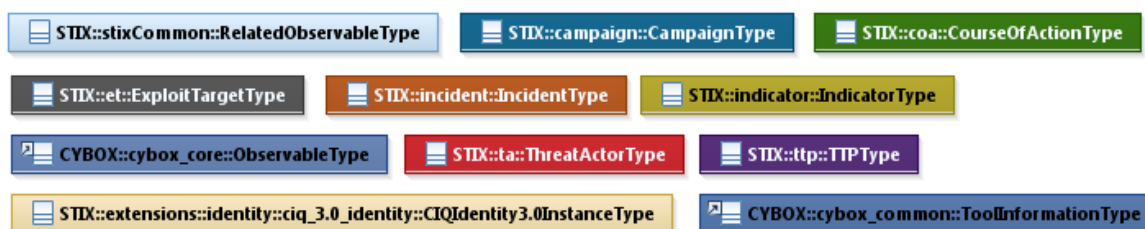


Figure 1-2. Data model color coding

### 1.2.4 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 Common 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\_Call<sub>A</sub>, Code<sub>B</sub>) and single logic expression in the Multiplicity column. For example, if there is a choice of property API\_Call<sub>A</sub> and Code<sub>B</sub>, 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.

### 1.2.5 Property and Class Descriptions

Each class and property defined in STIX is described using the format, “The X property verb Y.” For example, in the specification for the STIX Indicator, we write, “The id property specifies a globally unique identifier for the kill chain instance.” 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 STIX.

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

Verb	STIX Definition
<u>captures</u>	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.

	<p><i>Examples:</i></p> <p>The <code>Source</code> property characterizes the source of the sighting information. Examples of details <u>captured</u> include identifying characteristics, time-related attributes, and a list of the tools used to collect the information.</p> <p>The <code>Description</code> property <u>captures</u> a textual description of the Indicator.</p>
<u>characterizes</u>	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.
	<p><i>Examples:</i></p> <p>The <code>Confidence</code> property <u>characterizes</u> the level of confidence in the accuracy of the overall content captured in the Incident.</p> <p>The <code>ActivityType</code> class <u>characterizes</u> basic information about an activity a defender might use in response to a Campaign.</p>
<u>specifies</u>	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.
	<p><i>Example:</i></p> <p>The <code>version</code> property <u>specifies</u> the version identifier of the STIX Campaign data model used to capture the information associated with the Campaign.</p>

## 1.3 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 [\[RFC2119\]](#).

## 1.4 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>.
- [RFC4648]** Josefsson, S., “The Base16, Base32, and Base64 Data Encodings”, RFC 4648, October 2006. <https://tools.ietf.org/rfc/rfc4648.txt>

---

## 2 Background Information

In this section, we provide high level information about the Common data model that is useful to fully understand the Common data model specification details given in Section 3.

The STIX Common data model defines object classes that are shared across the various STIX data models. There is a wide variety of class types, so to make the specification document content easier to reference and understand, we have organized the data model content into eight categories:

- **Component Base Classes** – defined for each of the top-level STIX components<sup>2</sup>: Campaign, Course of Action, Exploit Target, Incident, Indicator, Threat Actor, TTP and Report.
- **Relationship-oriented Classes** – capture relationships between STIX constructs.
- **Content Aggregation Classes** – capture a collection of one or more STIX objects.
- **Kill Chain-related Classes** – facilitate the use of a phase-based model to describe the stages of an attack.
- **General Shared Classes** – serve a variety of purposes and shared by the collection of STIX data models.
- **General Data Types** – support the classes defined in the STIX data models.
- **Vocabulary Data Types** – provide a content creator with choices for defining content.
- **Enumerations** – support the classes defined in the STIX data models.

Each category is contained in a separate subsection in Section 3.

## 3 STIX<sup>™</sup> Common Data Model

There is no primary class of the STIX Common UML package because the Common data model contains a collection of classes that are used by the other STIX Packages. 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. In addition, in the Shared Classes section, if a set of classes are related in concept, they are also grouped and are not necessarily in alphabetical order. We list data types and enumerations in Sections 3.6 and 3.7, respectively.

### 3.1 Component Base Classes

The STIX Common data model provides base classes (superclasses) for each of the top-level STIX components<sup>3</sup>: Campaign, Course of Action, Exploit Target, Incident, Indicator, Threat Actor, TTP, and Report. The STIX Common base classes are minimal and are intended to be extended by the corresponding STIX component defined in that specification. The use of base classes allows the STIX language to be modular: all of the STIX components are defined in separate data models rather than in one large data model to limit interdependence between STIX components.

The default and strongly recommended class for fully implementing each STIX component is the primary class defined in the STIX component's data model. For example, consider the STIX Common `CampaignBaseType` base class, which is specialized in the Campaign data model to define the `CampaignType` class. The corresponding UML diagram is shown in Figure 3-1.

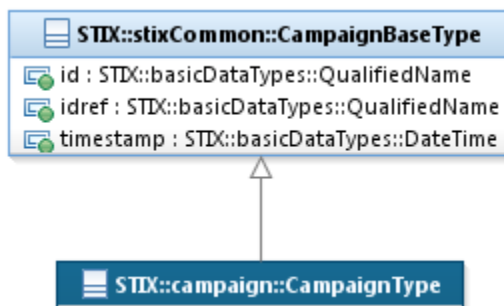


Figure 3-1. UML diagram showing the use of the `CampaignBaseType` base class

The collection of component base classes are defined in Sections 3.1.1 through 3.1.7; however, because all of the component superclasses are similar, UML diagrams analogous to Figure 3-1 are not included.

### 3.1.1 CampaignBaseType Class

The `CampaignBaseType` class is intended to be extended by a subclass, which characterizes a campaign. The decision to define base classes that are extended – like the `CampaignBaseType` class – was made to minimize interdependence between STIX components; it was not made to enable structural variation. The default and strongly RECOMMENDED subclass to extend the `CampaignBaseType` class is the `CampaignType` class in the Campaign data model (see [STIX Version 1.2.1 Part 8: Campaign](#)).

The property table of the `CampaignBaseType` base class is given in [Table 3-1](#).

Table 3-1. Properties of the `CampaignBaseType` base class

Name	Type	Multiplicity	Description
<b>id</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>id</code> property specifies a globally unique identifier for the Campaign instance.
<b>idref</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>idref</code> property specifies an identifier reference to a Campaign instance specified elsewhere. When the <code>idref</code> property is used, no properties other than <code>idref</code> and <code>timestamp</code> should be specified.
<b>timestamp</b>	<code>basicDataTypes:DateTime</code>	0..1	The <code>timestamp</code> property specifies the definition time of a specific version of a Campaign. When used in conjunction with the <code>id</code> property, it specifies the definition time for the specific version of the Campaign instance; when used in conjunction with the <code>idref</code> property, it specifies the definition time for a specific version of a Campaign instance defined elsewhere. The <code>timestamp</code> property has no defined semantic meaning if it is used without either the <code>id</code> or <code>idref</code> property.

### 3.1.2 CourseOfActionBaseType Class

The `CourseOfActionBaseType` class is intended to be extended by a subclass, which characterizes a course of action. The decision to define base classes that are extended – like the `CourseOfActionBaseType` class – was made to minimize interdependence between STIX components; it was not made to enable structural variation. The default and strongly RECOMMENDED subclass to extend the `CourseOfActionBaseType` class is the `CourseOfActionType` class in the Course of Action data model (see [STIX Version 1.2.1 Part 9: Course of Action](#)). The one case where the class SHOULD NOT be extended is when the `CourseOfActionBaseType` class is used as a reference via its `idref` property.

The property table of the `CourseOfActionBaseType` base class are given in [Table 3-2](#).

Table 3-2. Properties of the *CourseOfActionBaseType* base class

Name	Type	Multiplicity	Description
<b>id</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>id</code> property specifies a globally unique identifier for the Course of Action instance.
<b>idref</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>idref</code> property specifies an identifier reference to a Course of Action instance specified elsewhere. When the <code>idref</code> property is used, no properties other than <code>idref</code> and <code>timestamp</code> should be specified.
<b>timestamp</b>	<code>basicDataTypes:DateTime</code>	0..1	The <code>timestamp</code> property specifies the definition time of a specific version of a Course of Action. When used in conjunction with the <code>id</code> property, it specifies the definition time for the specific version of the Course of Action instance; when used in conjunction with the <code>idref</code> property, it specifies the definition time for a specific version of a Course of Action instance defined elsewhere. The <code>timestamp</code> property has no defined semantic meaning if it is used without either the <code>id</code> or <code>idref</code> property.

### 3.1.3 ExploitTargetBaseType Class

The `ExploitTargetBaseType` class is intended to be extended by a subclass, which characterizes an exploit target. The decision to define base classes that are extended – like the `ExploitTargetBaseType` class – was made to minimize interdependence between STIX components; it was not made to enable structural variation. The default and strongly RECOMMENDED subclass to extend the `ExploitTargetBaseType` class is the `ExploitTargetType` class in the Exploit Target data model (see [STIX Version 1.2.1 Part 10: Exploit Target](#)). The one case where the class SHOULD NOT be extended is when the `ExploitTargetBaseType` class is used as a reference via its `idref` property.

The property table of the `ExploitTargetBaseType` base class is given in [Table 3-3](#).

Table 3-3. Properties of the *ExploitTargetBaseType* base class

Name	Type	Multiplicity	Description
<b>id</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>id</code> property specifies a globally unique identifier for the Exploit Target instance.
<b>idref</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>idref</code> property specifies an identifier reference to an Exploit Target instance specified elsewhere. When the <code>idref</code> property is used,



			no properties other than <code>idref</code> and <code>timestamp</code> should be specified.
<b>timestamp</b>	<code>basicDataTypes:DateTime</code>	0..1	The <code>timestamp</code> property specifies the definition time of a specific version of an Exploit Target. When used in conjunction with the <code>id</code> property, it specifies the definition time for the specific version of the Exploit Target instance; when used in conjunction with the <code>idref</code> property, it specifies the definition time for a specific version of an Exploit Target instance defined elsewhere. The <code>timestamp</code> property has no defined semantic meaning if it is used without either the <code>id</code> or <code>idref</code> property.

### 3.1.4 IncidentBaseType Class

The `IncidentBaseType` class is intended to be extended by a subclass, which characterizes an incident. The decision to define base classes that are extended – like the `IncidentBaseType` class – was made to minimize interdependence between STIX components; it was not made to enable structural variation. The default and strongly RECOMMENDED subclass to extend the `IncidentBaseType` class is the `IncidentType` class in the Incident data model (see [STIX Version 1.2.1 Part 6: Incident](#)). The one case where the class SHOULD NOT be extended is when the `IncidentBaseType` class is used as a reference via its `idref` property.

The property table of the `IncidentBaseType` base class is given in [Table 3-4](#).

Table 3-4. Properties of the `IncidentBaseType` base class

Name	Type	Multiplicity	Description
<b>id</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>id</code> property specifies a globally unique identifier for the Incident instance.
<b>idref</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>idref</code> property specifies an identifier reference to an Incident instance specified elsewhere. When the <code>idref</code> property is used, no properties other than <code>idref</code> and <code>timestamp</code> should be specified.
<b>timestamp</b>	<code>basicDataTypes:DateTime</code>	0..1	The <code>timestamp</code> property specifies the definition time of a specific version of an Incident. When used in conjunction with the <code>id</code> property, it specifies the definition time for the specific version of the Incident instance; when used in conjunction with

			the <code>idref</code> property, it specifies the definition time for a specific version of an Incident instance defined elsewhere. The <code>timestamp</code> property has no defined semantic meaning if it is used without either the <code>id</code> or <code>idref</code> property.
--	--	--	--

### 3.1.5 IndicatorBaseType Class

The `IndicatorBaseType` class is intended to be extended by a subclass, which characterizes an indicator. The decision to define base classes that are extended – like the `IndicatorBaseType` class – was made to minimize interdependence between STIX components; it was not made to enable structural variation. The default and strongly RECOMMENDED subclass to extend the `IndicatorBaseType` class is the `IndicatorType` class in the Indicator data model (see [STIX Version 1.2.1 Part 4: Indicator](#)). The one case where the class SHOULD NOT be extended is when the `IndicatorBaseType` class is used as a reference via its `idref` property.

The property table of the `IndicatorBaseType` base class is given in [Table 3-5](#).

Table 3-5. Properties of the `IndicatorBaseType` base class

Name	Type	Multiplicity	Description
<b>id</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>id</code> property specifies a globally unique identifier for the Indicator instance.
<b>idref</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>idref</code> property specifies an identifier reference to an Indicator instance specified elsewhere. When the <code>idref</code> property is used, no properties other than <code>idref</code> and <code>timestamp</code> should be specified.
<b>timestamp</b>	<code>basicDataTypes:DateTime</code>	0..1	The <code>timestamp</code> property specifies the definition time of a specific version of an Indicator. When used in conjunction with the <code>id</code> property, it specifies the definition time for the specific version of the Indicator instance; when used in conjunction with the <code>idref</code> property, it specifies the definition time for a specific version of an Indicator instance defined elsewhere. The <code>timestamp</code> property has no defined semantic meaning if it is used without either the <code>id</code> or <code>idref</code> property.

### 3.1.6 ThreatActorBaseType Class

The `ThreatActorBaseType` class is intended to be extended by a subclass, which characterizes a threat actor. The decision to define base classes that are extended – like the `ThreatActorBaseType` class – was made to minimize interdependence between STIX components; it was not made to enable structural variation. The default and strongly RECOMMENDED subclass to extend the `ThreatActorBaseType` class is the `ThreatActorType` class in the Threat Actor data model (see [STIX Version 1.2.1 Part 7: Threat Actor](#)). The one case where the class SHOULD NOT be extended is when the `ThreatActorBaseType` class is used as a reference via its `idref` property.

The property table of the `ThreatActorBaseType` base class is given in [Table 3-6](#).

Table 3-6. Properties of the `ThreatActorBaseType` base class

Name	Type	Multiplicity	Description
<b>id</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>id</code> property specifies a globally unique identifier for the Threat Actor instance.
<b>idref</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>idref</code> property specifies an identifier reference to a Threat Actor instance specified elsewhere. When the <code>idref</code> property is used, no properties other than <code>idref</code> and <code>timestamp</code> should be specified.
<b>timestamp</b>	<code>basicDataTypes:DateTime</code>	0..1	The <code>timestamp</code> property specifies the definition time of a specific version of a Threat Actor. When used in conjunction with the <code>id</code> property, it specifies the definition time for the specific version of the Threat Actor instance; when used in conjunction with the <code>idref</code> property, it specifies the definition time for a specific version of a Threat Actor instance defined elsewhere. The <code>timestamp</code> property has no defined semantic meaning if it is used without either the <code>id</code> or <code>idref</code> property.

### 3.1.7 TTPBaseType Class

The `TTPBaseType` class is intended to be extended by a subclass, which characterizes a TTP. The decision to define base classes that are extended – like the `TTPBaseType` class – was made to minimize interdependence between STIX components; it was not made to enable structural variation. The default and strongly RECOMMENDED subclass to extend the `TTPBaseType` class is the `TTPType` class in the TTP data model (see [STIX Version 1.2.1 Part 5: TTP](#)). The one case where the class SHOULD NOT be extended is when the `TTPBaseType` class is used as a reference via its `idref` property.

The property table of the `TTPBaseType` base class is given in [Table 3-7](#).

Table 3-7. Properties of the *TTPBaseType* base class

Name	Type	Multiplicity	Description
<b>id</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>id</code> property specifies a globally unique identifier for the TTP instance.
<b>idref</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>idref</code> property specifies an identifier reference to a TTP instance specified elsewhere. When the <code>idref</code> property is used, no properties other than <code>idref</code> and <code>timestamp</code> should be specified.
<b>Timestamp</b>	<code>basicDataTypes:DateTime</code>	0..1	The <code>timestamp</code> property specifies the definition time of a specific version of a TTP. When used in conjunction with the <code>id</code> property, it specifies the definition time for the specific version of the TTP instance; when used in conjunction with the <code>idref</code> property, it specifies the definition time for a specific version of a TTP instance defined elsewhere. The <code>timestamp</code> property has no defined semantic meaning if it is used without either the <code>id</code> or <code>idref</code> property.

### 3.1.8 ReportBaseType Class

The `ReportBaseType` class is intended to be extended by a subclass, which characterizes a report. The decision to define base classes that are extended – like the `ReportBaseType` class – was made to minimize interdependence between STIX components; it was not made to enable structural variation. The default and strongly RECOMMENDED subclass to extend the `ReportBaseType` class is the `ReportType` class in the Report data model (see [STIX Version 1.2.1 Part 11: Report](#)). The one case where the class SHOULD NOT be extended is when the `ReportBaseType` class is used as a reference via its `idref` property.

The property table of the `ReportBaseType` base class is given in [Table 3-8](#).

Table 3-8. Properties of the *ReportBaseType* base class

Name	Type	Multiplicity	Description
<b>id</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>id</code> property specifies a globally unique identifier for the Report.
<b>idref</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>idref</code> property specifies an identifier reference to a Report specified elsewhere. When the <code>idref</code> property is used, no properties other than <code>idref</code> and <code>timestamp</code> should be specified.

<b>timestamp</b>	<code>basicDataTypes:DateTime</code>	0..1	When used in conjunction with the <code>id</code> property, the <code>timestamp</code> property specifies the definition time for the specific version of the Report; when used in conjunction with the <code>idref</code> property, it specifies the definition time for a specific version of a Report defined elsewhere. The <code>timestamp</code> property has no defined semantic meaning if it is used without either the <code>id</code> or <code>idref</code> property.
------------------	--------------------------------------	------	--

## 3.2 Relationship-Oriented Classes

The STIX Common data model defines a number of classes that capture relationships between STIX constructs through specialization of the `GenericRelationshipType` abstract class, which is defined first.

### 3.2.1 GenericRelationshipType Class

The `GenericRelationshipType` class captures attributes associated with a relationship between a subject STIX construct and another STIX construct. It is an abstract class, and it **MUST** be extended via a subclass to specify the other STIX constructs. Use of the `GenericRelationshipType` class helps to define a consistent structure for relationships that go beyond a simple statement of “construct A is related to construct B.” Instead, additional properties of the relationship can be specified such as the nature of the relationship and the level of confidence in the presence and accuracy of the asserted relationship.

The property table of the `GenericRelationshipType` class is given in [Table 3-9](#).

Table 3-9. Properties of the `GenericRelationshipType` class

Name	Type	Multiplicity	Description
<b>Confidence</b>	<code>ConfidenceType</code>	0..1	The <code>Confidence</code> property characterizes the level of confidence in the presence and accuracy of the asserted relationship.
<b>Information_Source</b>	<code>InformationSourceType</code>	0..1	The <code>Information_Source</code> property characterizes the source of the relationship information. Examples of details captured include identifying characteristics, time-related attributes, and a list of tools used to collect the information.
<b>Relationship</b>	<code>VocabularyStringType</code>	0..1	The <code>Relationship</code> property specifies the nature of the relationship between two STIX constructs. Examples of potential natures of relationship include <i>Updates - Revises</i> (for versioning of content), <i>Asserted Same</i> (for relating two Threat Actors), and <i>Leverages</i> (for relating two TTPs). 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 <code>ControlledVocabularyStringType</code> class. No default vocabulary is defined in STIX v1.2.1.
--	--	--	---

As an example of how the `GenericRelationshipType` class is used, consider the UML diagram shown in [Figure 3-2](#), which shows the relationship between an Indicator and a TTP. As illustrated, the `GenericRelationshipType` class enables capture of information that further defines the relationship between the Indicator and the TTP. Namely, it specifies the level of confidence that the two constructs are related and the source of the relationship information, and it characterizes the nature of the relationship between the two constructs.

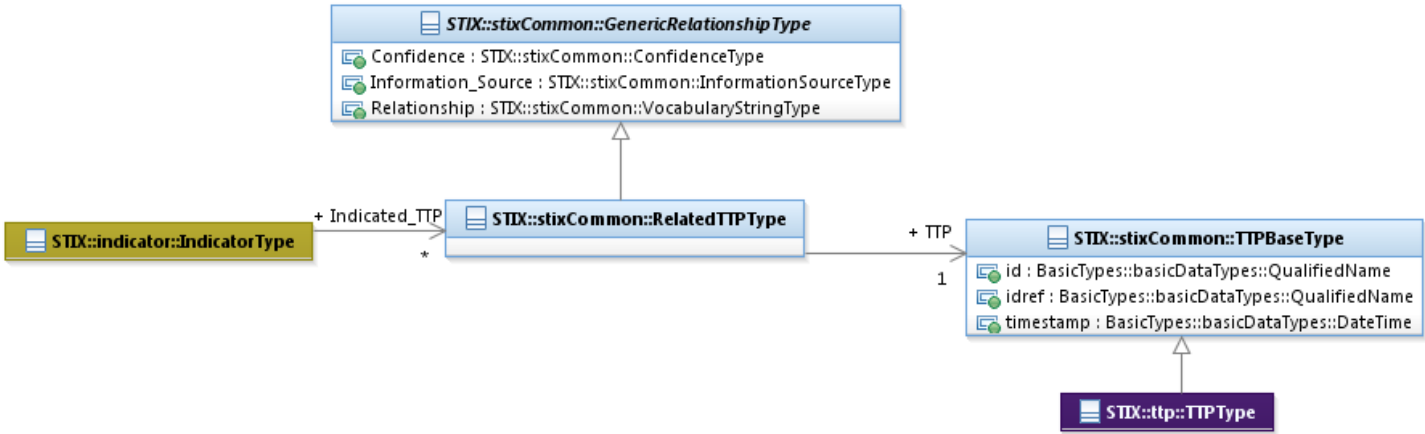


Figure 3-2. UML diagram of the Indicator/TTP relationship

### 3.2.2 RelatedCampaignType Class

The `RelatedCampaignType` class characterizes a relationship to a Campaign. It extends the `GenericRelationshipType` superclass by specifying the Campaign.

The UML diagram corresponding to the `RelatedCampaignType` class is shown in [Figure 3-3](#).

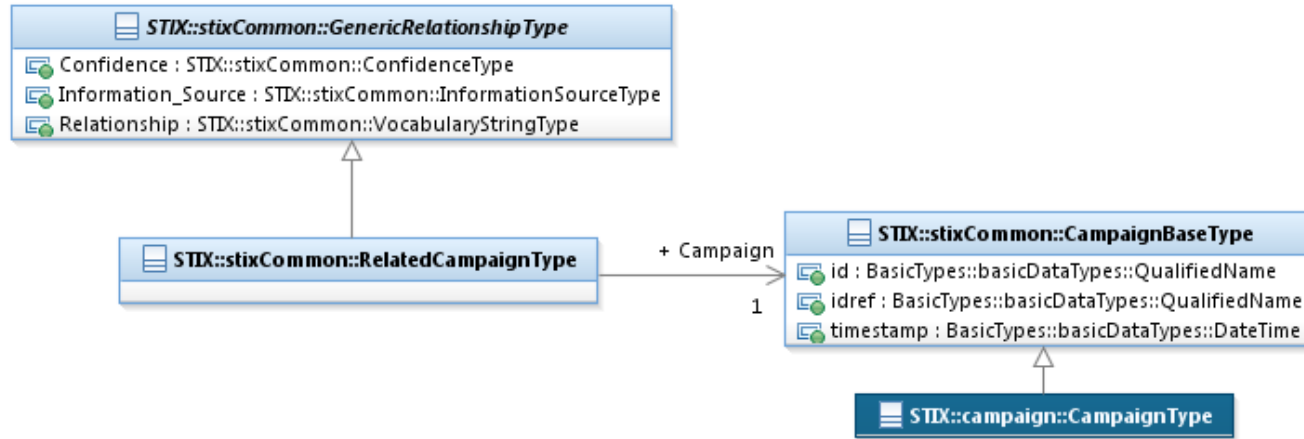


Figure 3-3. UML diagram of the *RelatedCampaignType* class

The property table given in [Table 3-10](#) corresponds to the UML diagram shown in [Figure 3-3](#).

Table 3-10. Properties of the *RelatedCampaignType* class

Name	Type	Multiplicity	Description
<b>Campaign</b>	CampaignBaseType	1	The Campaign property characterizes a cyber threat Campaign. The CampaignBaseType class is a minimal base class that is intended to be extended. The default and strongly RECOMMENDED class to fully implement a Campaign is the campaign:CampaignType class defined in <a href="#">STIX Version 1.2.1 Part 8: Campaign</a> . Base classes are used to minimize interdependence between STIX components, not to enable or encourage conflicting syntactic variation. Through the use of the idref property, a reference to a Campaign defined elsewhere can be specified via the direct use of the CampaignBaseType class.

### 3.2.3 RelatedCampaignReferenceType Class

The RelatedCampaignReferenceType class characterizes a relationship to a Campaign. It extends the GenericRelationshipType superclass by specifying a reference to the Campaign. Unlike most other relationships that are defined in STIX, the RelatedCampaignReferencesType class does not allow a Campaign to be embedded; an already-defined Campaign **MUST** be specified by its Name property and/or a reference to its identifier.

The UML diagram corresponding to the `RelatedCampaignReferenceType` class is shown in [Figure 3-4](#).

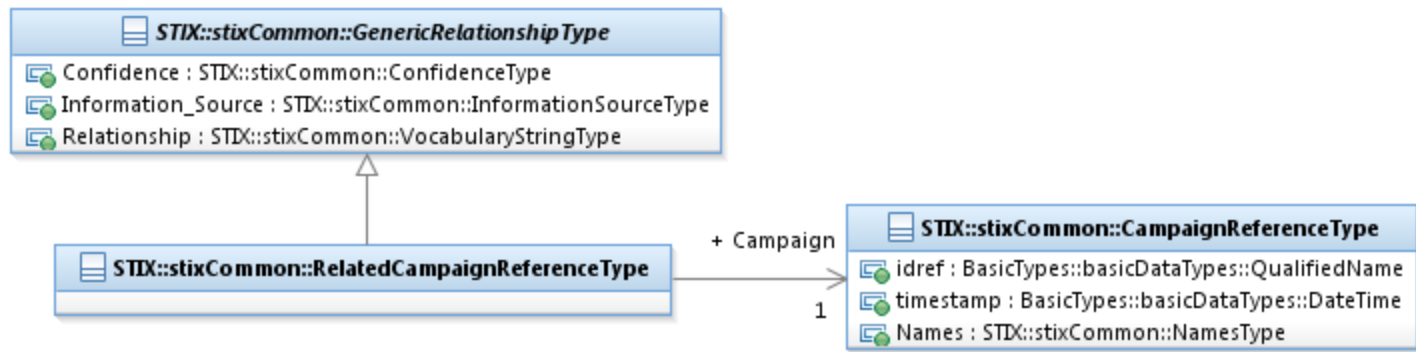


Figure 3-4. UML diagram of the `RelatedCampaignReferenceType` class

The property table given in [Table 3-11](#) corresponds to the UML diagram shown in [Figure 3-4](#).

Table 3-11. Properties of the `RelatedCampaignReferenceType` class

Name	Type	Multiplicity	Description
Campaign	CampaignReferenceType	0..1	The Campaign property captures a reference to the related Campaign.

The `RelatedCampaignReferenceType` class is defined in addition to the `RelatedCampaignType` class because in some cases it is not appropriate to define a Campaign *only* in the context of another construct, and in those cases, an otherwise defined Campaign should be referenced. For example, a Campaign should not be defined only in the context of a single Indicator because the Indicator may be used across many Campaigns; a relationship between an Indicator and a Campaign should be a reference-only relationship (see the `Related_Campaigns` class defined in [STIX Version 1.2.1 Part 4: Indicator](#).

### 3.2.3.1 CampaignReferenceType Class

The `CampaignReferenceType` class captures a reference to a related Campaign.

The property table of the `CampaignReferenceType` class is given in [Table 3-12](#).

Table 3-12. Properties of the `CampaignReferenceType` class



Name	Type	Multiplicity	Description
<b>idref</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>idref</code> property specifies a globally unique identifier for a Campaign defined elsewhere.
<b>timestamp</b>	<code>basicDataTypes:DateTime</code>	0..1	The <code>timestamp</code> property, used in combination with the <code>idref</code> property, specifies a specific version of a Campaign defined elsewhere. To avoid ambiguity, all timestamps SHOULD include a specification of the time zone.
<b>Names</b>	<code>NamesType</code>	0..1	The <code>Names</code> property specifies a set of one or more names (i.e., aliases) used to identify a referenced Campaign (specified by an <code>idref</code> property). An organization may use names that are created internally or externally (outside the organization).

### 3.2.4 RelatedCourseOfActionType Class

The `RelatedCourseOfActionType` class characterizes a relationship to a Course of Action. It extends the `GenericRelationshipType` superclass by specifying a related Course of Action.

The UML diagram corresponding to the `RelatedCourseOfActionType` class is shown in [Figure 3-5](#).

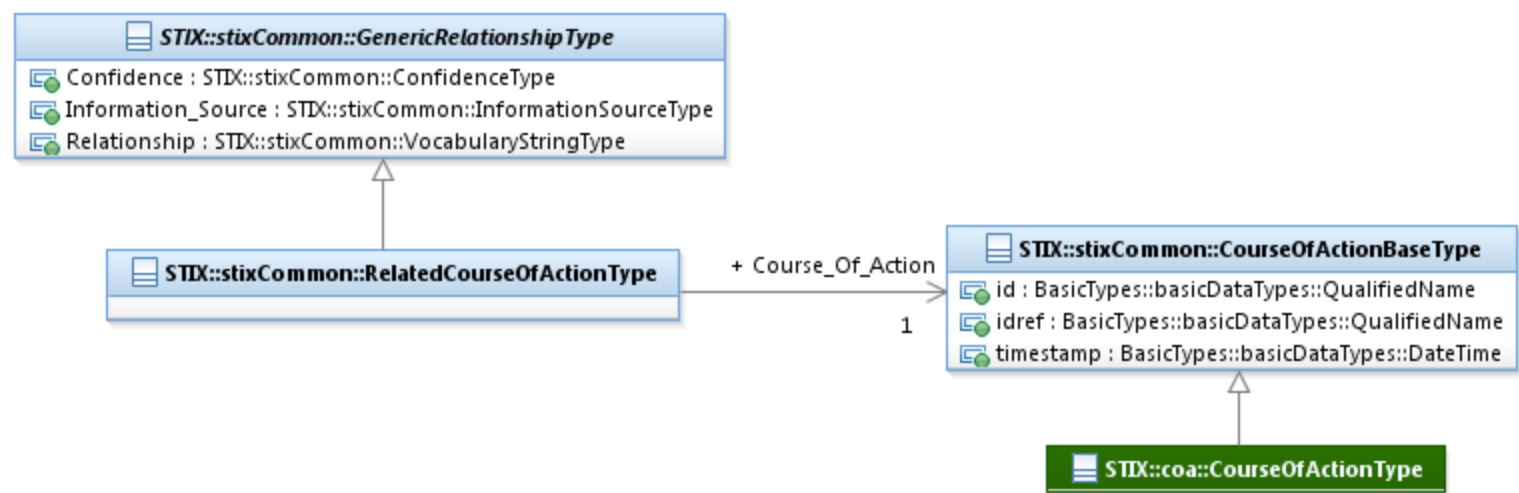


Figure 3-5. UML diagram of the `RelatedCourseOfActionType` class

The property table given in [Table 3-13](#) corresponds to the UML diagram shown in [Figure 3-5](#).

Table 3-13. Properties of the *RelatedCourseOfActionType* class

Name	Type	Multiplicity	Description
<b>Course_Of_Action</b>	CourseOfActionBaseType	1	The <code>Course_Of_Action</code> property characterizes a Course of Action that could be taken in regard to one of more cyber threats. The <code>CourseOfActionBaseType</code> class is a minimal base class that is intended to be extended. The default and strongly RECOMMENDED class to fully implement a Course of Action is the <code>coa:CourseOfActionType</code> class defined in <a href="#">STIX Version 1.2.1 Part 9: Course of Action</a> . Base classes are used to minimize interdependence between STIX components, not to enable or encourage conflicting syntactic variation. Through the use of the <code>idref</code> property, a reference to a Course of Action defined elsewhere can be specified via the direct use of the <code>CourseOfActionBaseType</code> class.

### 3.2.5 RelatedExploitTargetType Class

The `RelatedExploitTargetType` class characterizes a relationship to an Exploit Target. It extends the `GenericRelationshipType` superclass by specifying a related Exploit Target.

The UML diagram corresponding to the `RelatedExploitTargetType` class is shown in [Figure 3-6](#).

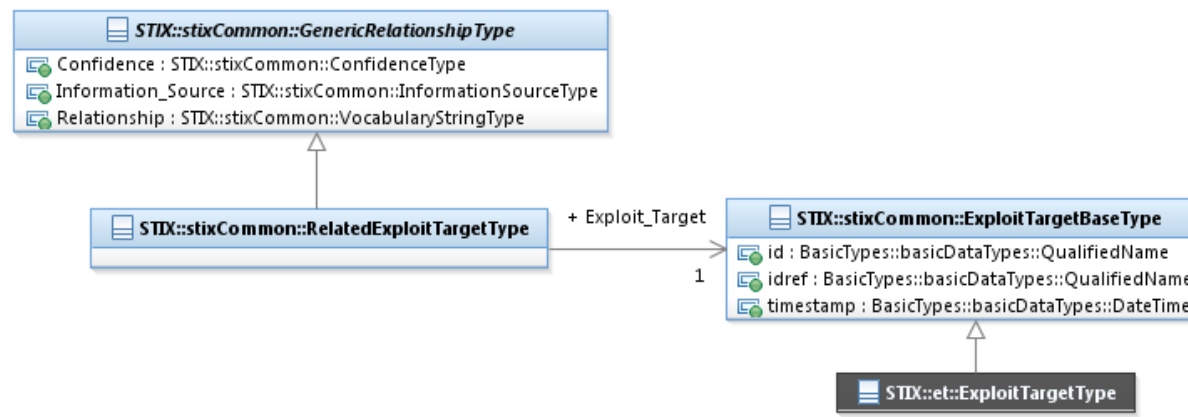


Figure 3-6. UML diagram of the *RelatedExploitTargetType* class

The property table given in [Table 3-14](#) corresponds to the UML diagram shown in [Figure 3-6](#).

Table 3-14. Properties of the *RelatedExploitTargetType* class

Name	Type	Multiplicity	Description
<b>Exploit_Target</b>	ExploitTargetBaseType	1	The <code>Exploit_Target</code> property characterizes an Exploit Target. The <code>ExploitTargetBaseType</code> class is a minimal base class that is intended to be extended. The default and strongly RECOMMENDED class to fully implement an Exploit Target is the <code>et:ExploitTargetType</code> class defined in <a href="#">STIX Version 1.2.1 Part 10: Exploit Target</a> . Base classes are used to minimize interdependence between STIX components, not to enable or encourage conflicting syntactic variation. Through the use of the <code>idref</code> property, a reference to an Exploit Target defined elsewhere can be specified via the direct use of the <code>ExploitTargetBaseType</code> class.

### 3.2.6 RelatedIdentityType Class

The `RelatedIdentityType` class characterizes a relationship to an identity. It extends the `GenericRelationshipType` superclass by specifying a related Identity.

The UML diagram corresponding to the `RelatedIdentityType` class is shown in [Figure 3-7](#).

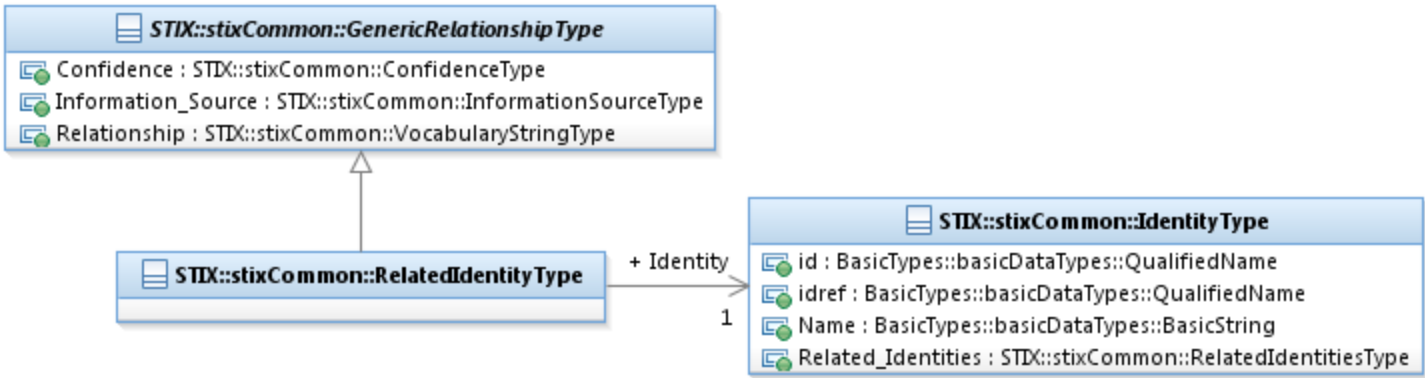


Figure 3-7. UML diagram of the `RelatedIdentityType` class

The property table given in [Table 3-15](#) corresponds to the UML diagram shown in [Figure 3-7](#).

Table 3-15. Properties of the `RelatedIdentityType` class

Name	Type	Multiplicity	Description
Identity	IdentityType	1	The <code>Identity</code> property characterizes the related identity. For situations calling for more than a simple name, the underlying class may be extended using a more complete structure such as the <code>CIQIdentity3.0InstanceType</code> subclass as defined <a href="#">STIX Version 1.2.1 Part 12: Default Extensions</a> . Through the use of the <code>idref</code> property, a reference to an <code>Identity</code> defined elsewhere can be specified via the direct use of the <code>IdentityType</code> class.

### 3.2.7 RelatedIncidentType Class

The `RelatedIncidentType` class characterizes a relationship to an `Incident`. It extends the `GenericRelationshipType` superclass by specifying a related `Incident`.

The UML diagram corresponding to the `RelatedIncidentType` class is shown in [Figure 3-8](#).

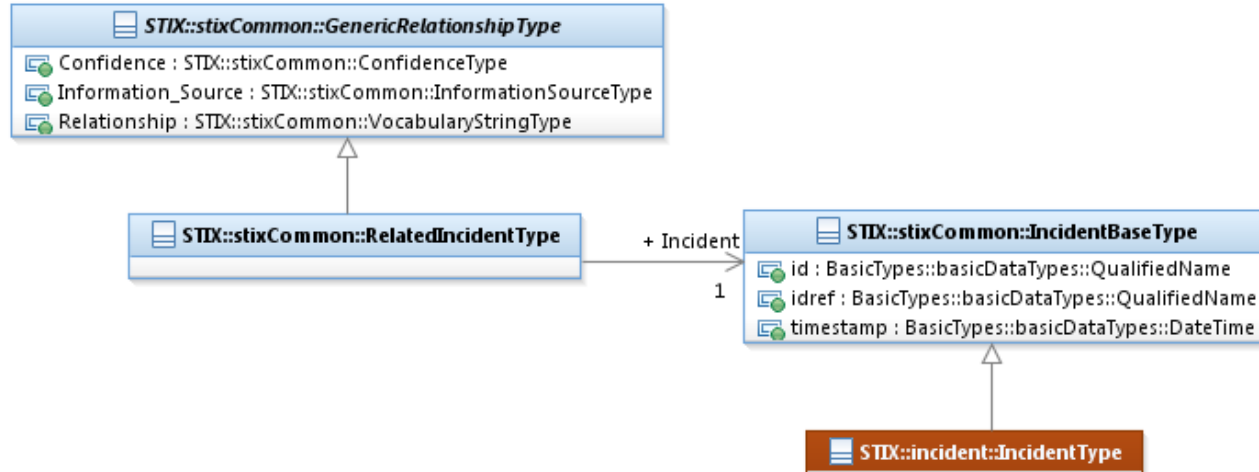


Figure 3-8. UML diagram of the *RelatedIncidentType* class

The property table given in [Table 3-16](#) corresponds to the UML diagram shown in [Figure 3-8](#).

Table 3-16. Properties of the *RelatedIncidentType* class

Name	Type	Multiplicity	Description
<b>Incident</b>	IncidentBaseType	1	The <code>Incident</code> property characterizes a cyber threat Incident. The <code>IncidentBaseType</code> class is a minimal base class that is intended to be extended. The default and strongly RECOMMENDED class to fully implement an Incident is the <code>incident:IncidentType</code> class defined in <a href="#">STIX Version 1.2.1 Part 6: Incident</a> . Base classes are used to minimize interdependence between STIX components, not to enable or encourage conflicting syntactic variation. Through the use of the <code>idref</code> property, a reference to an Incident defined elsewhere can be specified via the direct use of the <code>IncidentBaseType</code> class.

### 3.2.8 RelatedIndicatorType Class

The `RelatedIndicatorType` class characterizes a relationship to an Indicator. It extends the `GenericRelationshipType` superclass by specifying a related Indicator.

The UML diagram corresponding to the `RelatedIndicatorType` class is shown in [Figure 3-9](#).

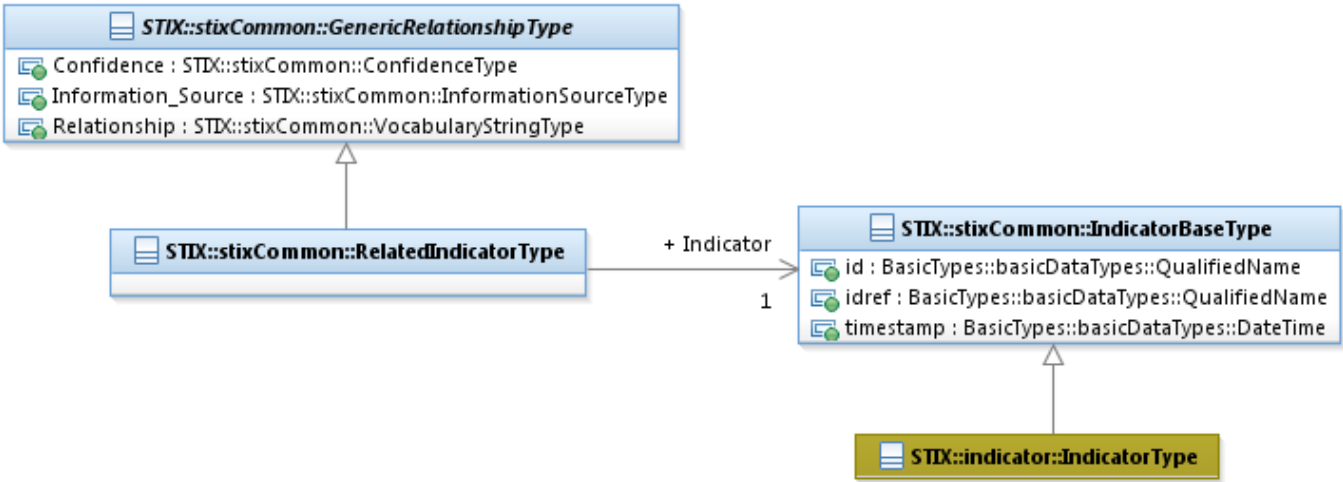


Figure 3-9. UML diagram of the `RelatedIndicatorType` class

The property table given in [Table 3-17](#) corresponds to the UML diagram shown in [Figure 3-9](#).

Table 3-17. Properties of the *RelatedIndicatorType* class

Name	Type	Multiplicity	Description
<b>Indicator</b>	<code>IndicatorBaseType</code>	1	The <code>Indicator</code> property characterizes a cyber threat Indicator. The <code>IndicatorBaseType</code> class is a minimal base class that is intended to be extended. The default and strongly RECOMMENDED class to fully implement an Indicator is the <code>indicator:IndicatorType</code> class defined in <a href="#">STIX Version 1.2.1 Part 4: Indicator</a> . Base classes are used to minimize interdependence between STIX components, not to enable or encourage conflicting syntactic variation. Through the use of the <code>idref</code> property, a reference to an Indicator defined elsewhere can be specified via the direct use of the <code>IndicatorBaseType</code> class.

### 3.2.9 RelatedObservableType Class

The `RelatedObservableType` class characterizes a relationship to a CybOX Observable. It extends the `GenericRelationshipType` superclass by specifying a related Observable.

The UML diagram corresponding to the `RelatedObservableType` class is shown in [Figure 3-10](#).

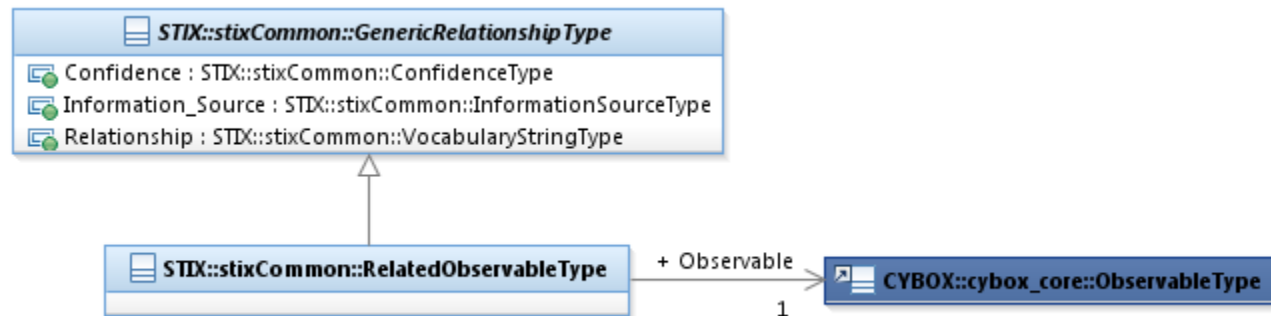


Figure 3-10. UML diagram of the *RelatedObservableType* class

The property table given in [Table 3-18](#) corresponds to the UML diagram shown in [Figure 3-10](#).



Table 3-18. Properties of the *RelatedObservableType* class

Name	Type	Multiplicity	Description
<b>Observable</b>	<code>cybox_core:ObservableType</code>	1	The <code>Observable</code> property characterizes the related cyber observable.

### 3.2.10 RelatedPackageRefType Class

The `RelatedPackageRefType` class characterizes a relationship to a STIX Package. It extends the `GenericRelationshipType` superclass by specifying the Package.

Because it would not make sense to define a totally new STIX package in the context of any single STIX component, all relationships between a STIX Package and a component are reference type relationships using the `RelatedPackageRefType` class (i.e., a `RelatedPackageType` class is not defined).

The UML diagram associated with the `RelatedPackageRefType` class is shown in [Figure 3-11](#).

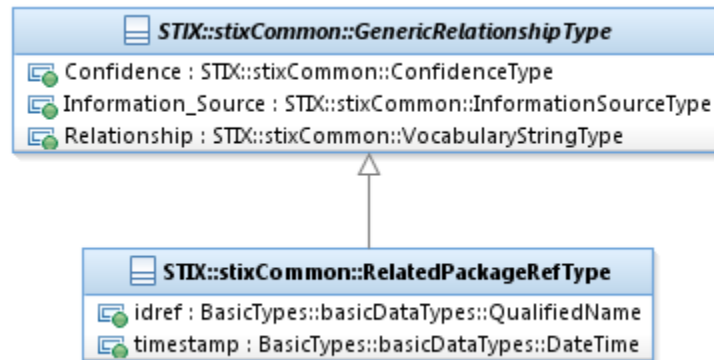


Figure 3-11. UML diagram of the *RelatedPackageRefType* class

The property table given in [Table 3-19](#) corresponds to the UML diagram shown in [Figure 3-11](#).

Table 3-19. Properties of the *RelatedPackageRefType* class

Name	Type	Multiplicity	Description
<b>idref</b>	<code>basicDataTypes:QualifiedNames</code>	0..1	The <code>idref</code> property specifies a globally unique identifier of a STIX Package specified elsewhere.
<b>timestamp</b>	<code>basicDataTypes:DateTime</code>	0..1	The <code>timestamp</code> property, used in combination with the <code>idref</code> property, specifies a specific version of a Package defined elsewhere. To avoid ambiguity, all timestamps SHOULD include a specification of the time zone.

### 3.2.11 RelatedReportType Class

The `RelatedReportType` class characterizes a relationship to a Report. It extends the `GenericRelationshipType` superclass by specifying a related Report.

The UML diagram corresponding to the `RelatedReportType` class is shown in [Figure 3-12](#).

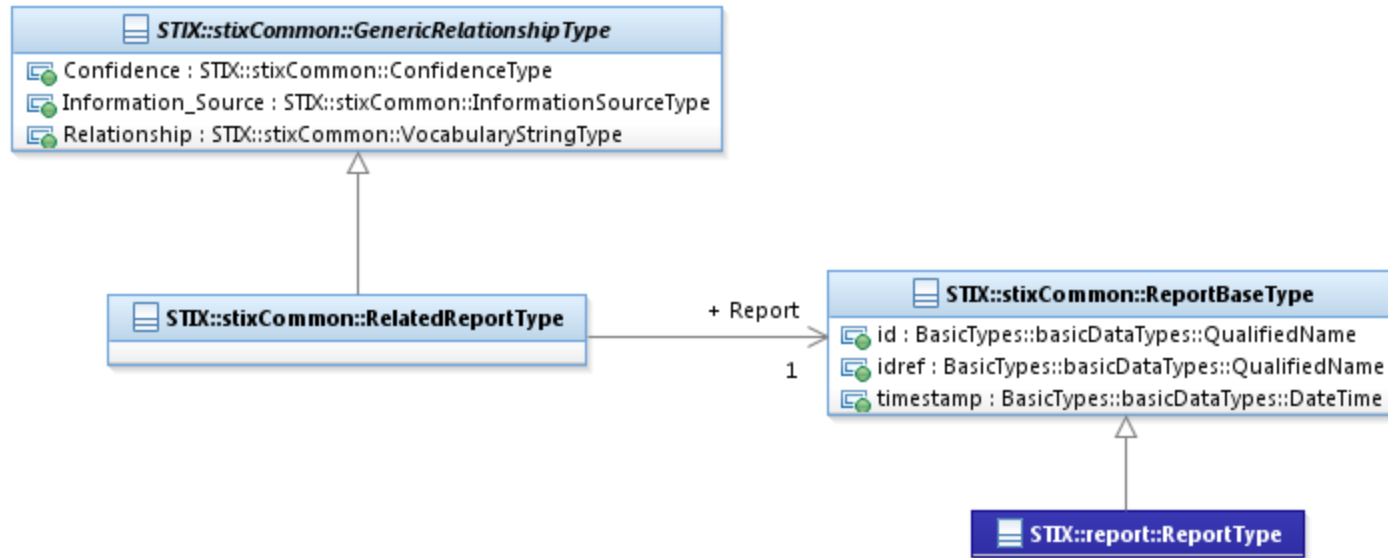


Figure 3-12. UML diagram of the *RelatedReportType* class

The property table given in [Table 3-20](#) corresponds to the UML diagram shown in [Figure 3-12](#).

Table 3-20. Properties of the *RelatedThreatActorType* class

Name	Type	Multiplicity	Description
<b>Report</b>	ReportBaseType	1	The Report property characterizes a STIX Report. The ReportBaseType class is a minimal base class that is intended to be extended. The default and strongly RECOMMENDED class to fully implement a Report is the report:ReportType class defined in <a href="#">STIX Version 1.2.1 Part 11: Report</a> . Base classes are used to minimize interdependence between STIX components, not to enable or encourage conflicting syntactic variation. Through the use of the idref property, a reference to a Report defined elsewhere can be specified via the direct use of the ReportBaseType class.

3.2.12 RelatedThreatActorType Class

The RelatedThreatActorType class characterizes a relationship to a Threat Actor. It extends the GenericRelationshipType superclass by specifying a related Threat Actor.

The UML diagram corresponding to the RelatedThreatActorType class is shown in Figure 3-13.

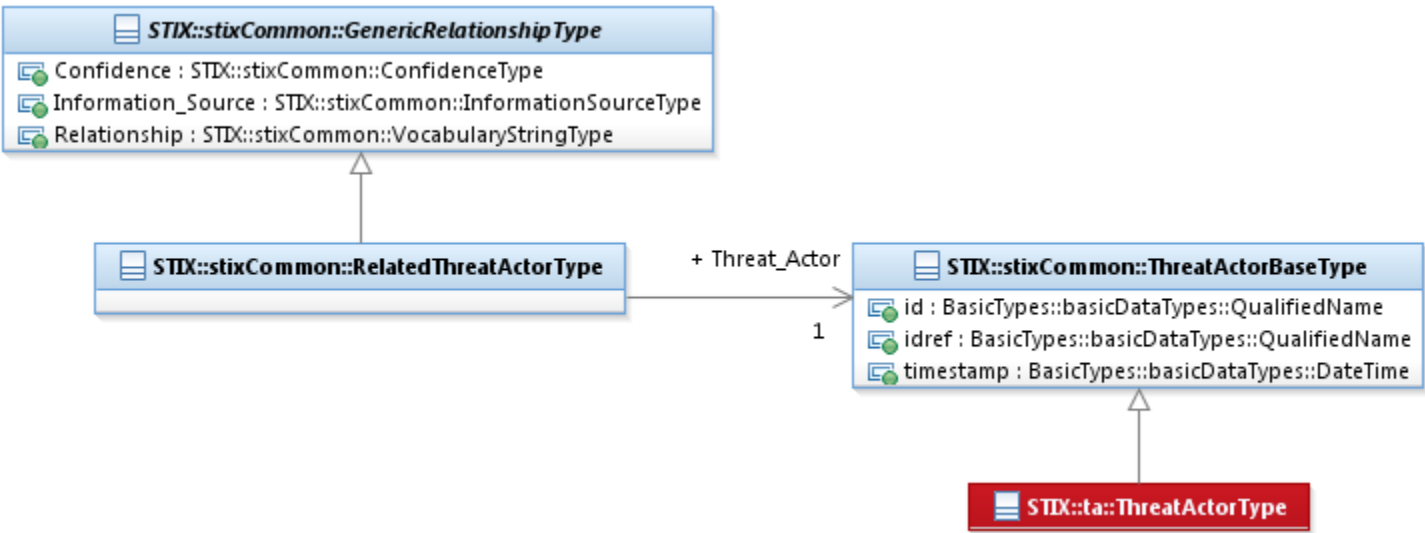


Figure 3-13. UML diagram of the RelatedThreatActorType class

The property table given in Table 3-21 corresponds to the UML diagram shown in Figure 3-13.

Table 3-21. Properties of the RelatedThreatActorType class

Name	Type	Multiplicity	Description
Threat_Actor	ThreatActorBaseType	1	The ThreatActor property characterizes a cyber Threat Actor. The ThreatActorBaseType class is a minimal base class that is intended to be extended. The default and strongly RECOMMENDED class to fully implement an ThreatActor is the ta:ThreatActorType class defined in <a href="#">STIX Version 1.2.1 Part 7: Threat Actor</a> . Base classes are used to

			minimize interdependence between STIX components, not to enable or encourage conflicting syntactic variation. Through the use of the <code>idref</code> property, a reference to a Threat Actor defined elsewhere can be specified via the direct use of the <code>ThreatActorBaseType</code> class.
--	--	--	--

### 3.2.13 RelatedTTPType Class

The `RelatedTTPType` class characterizes a relationship to a TTP. It extends the `GenericRelationshipType` superclass by specifying a related TTP.

The UML diagram corresponding to the `RelatedTTPType` class is shown in [Figure 3-14](#).

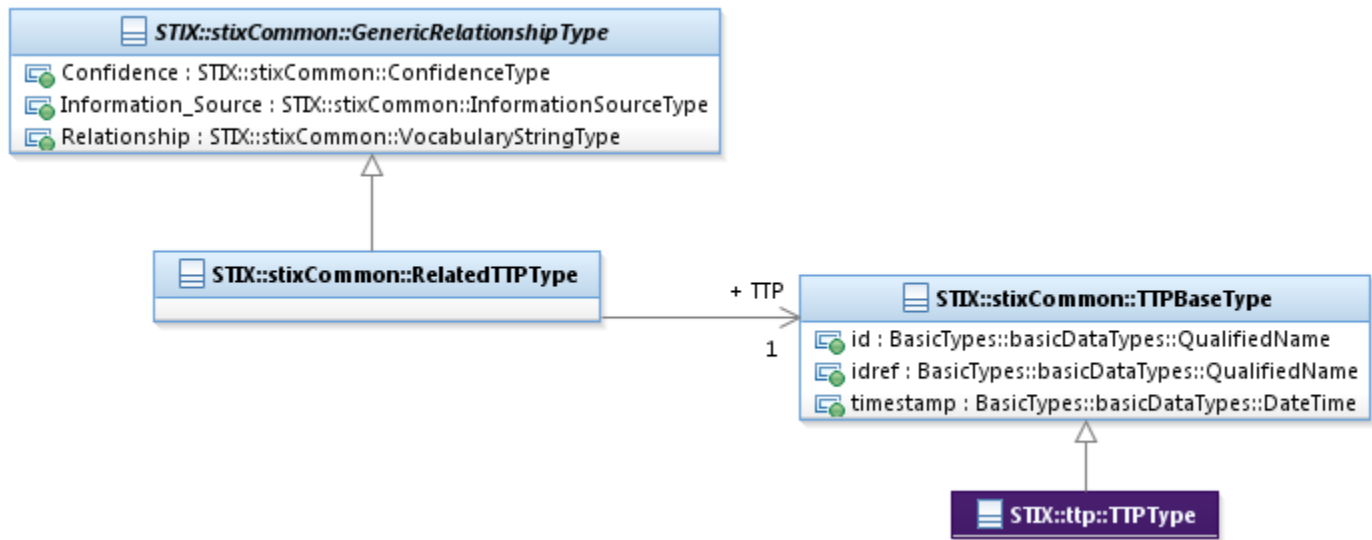


Figure 3-14. UML diagram of the `RelatedTTPType` class

The property table given in [Table 3-22](#) corresponds to the UML diagram shown in [Figure 3-14](#).

Table 3-22. Properties of the *RelatedTTPType* class

Name	Type	Multiplicity	Description
TTP	TTPBaseType	1	The TTP property characterizes a cyber threat adversary Tactic, Technique or Procedure (TTP). The <code>TTPBaseType</code> class is a minimal base class that is intended to be extended. The default and strongly RECOMMENDED class to fully implement a TTP is the <code>ttp:TTPType</code> class defined in <a href="#">STIX Version 1.2.1 Part 5: TTP</a> . Base classes are used to minimize interdependence between STIX components, not to enable or encourage conflicting syntactic variation. Through the use of the <code>idref</code> property, a reference to a TTP defined elsewhere can be specified via the direct use of the <code>TTPBaseType</code> class.

### 3.3 Content Aggregation Classes

A content aggregation class captures a collection of one (or zero, in some cases) or more STIX objects. Some content aggregation classes are very straightforward and simply capture a set of objects. However, others such as the `GenericRelationshipListType` abstract class are intended to be extended (see Section 3.3.1).

#### 3.3.1 GenericRelationshipListType

The `GenericRelationshipListType` class specifies how the relationships between a subject STIX construct and a set of one or more other STIX constructs should be interpreted. It is an abstract class, and it MUST be extended via a subclass to characterize the actual set of related constructs.

Classes that extend the `GenericRelationshipListType` class are used to *explicitly* represent the one-to-many relationship between constructs. In this way, the model enables the ability to express a property of the set itself. Currently, the only property defined is the *scope* of the relationship – whether the elements of the set are related individually or as a group.

Examples of STIX classes that extend the `GenericRelationshipListType` class include the `AttributeType` class defined in the Campaign data model, the `RelatedIndicatorsType` class in the Incident data model, and the `ExploitTargetsType` class defined in the TTP data model (to name just a few). An explicit example is given in [Figure 3-15](#), which illustrates how the `GenericRelationshipListType` class can be extended by the `RelatedIndicatorsType` class to associate a set of Indicators to an Incident.

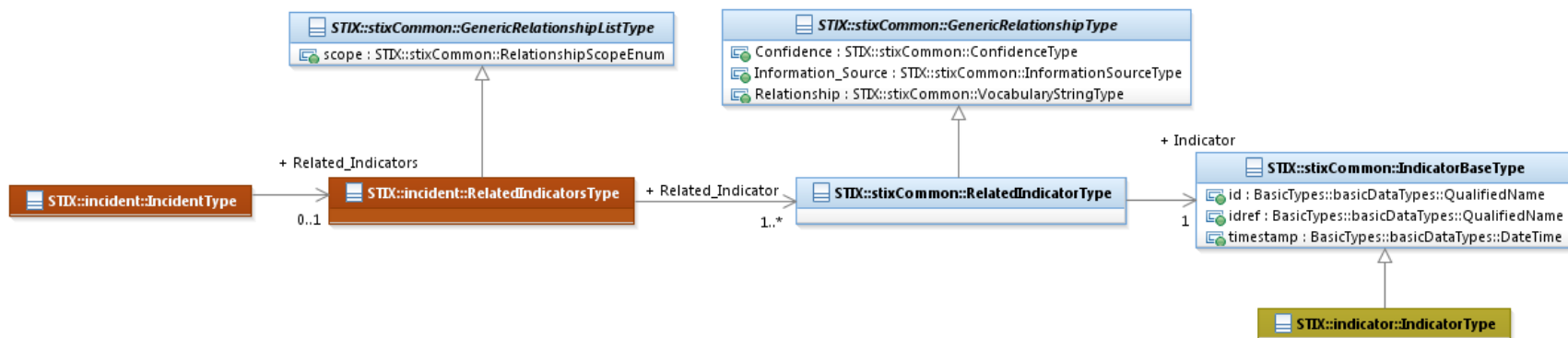


Figure 3-15. Example extension of the *GenericRelationshipListType* class

The property table given in [Table 3-23](#) corresponds to the UML diagram shown in [Figure 3-15](#).

Table 3-23. Properties of the *GenericRelationshipListType* class

Name	Type	Multiplicity	Description
<b>scope</b>	RelationshipScopeEnum	0..1	The <i>scope</i> property specifies how the set of relationships should be interpreted. Potential values are specified by the <i>ScopeEnum</i> enumeration. If ' <i>inclusive</i> ' is specified, then a single conceptual relationship is being defined between the subject construct and the combined collection of related constructs. If ' <i>exclusive</i> ' is specified (the default), then multiple relationships are being defined between the subject construct and each individual related construct.

### 3.3.2 ConfidenceAssertionChainType Class

The *ConfidenceAssertionChainType* class specifies a set of one or more related confidence levels in an assertion.

The property table of the *ConfidenceAssertionChainType* class is given in [Table 3-24](#).

Table 3-24. Properties of the *ConfidenceAssertionChainType* class

Name	Type	Multiplicity	Description
<b>Confidence_Assertion</b>	ConfidenceType	1..*	The <i>Confidence_Assertion</i> property characterizes confidence in an assertion.

### 3.3.3 ContributingSourcesType Class

The *ContributingSourcesType* class specifies a set of one or more information sources contributing to the data entry.

The UML diagram corresponding to the *ContributingSourcesType* class is shown in [Figure 3-16](#).

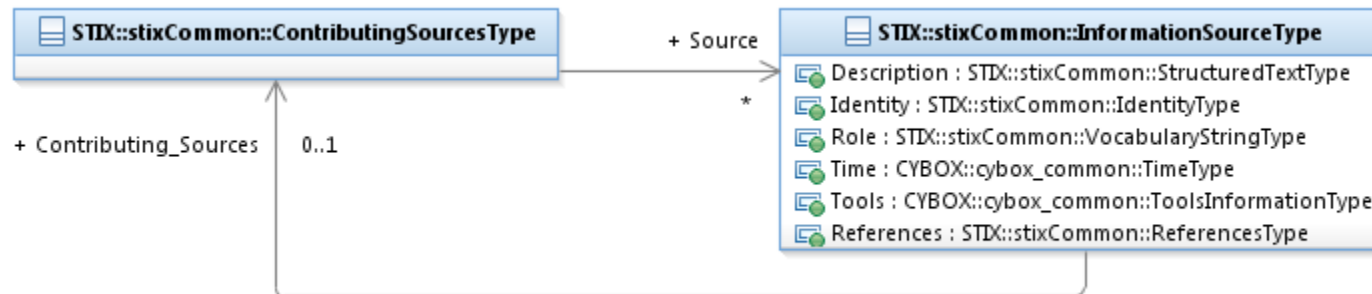


Figure 3-16. UML diagram of the *ContributingSourcesType* class

As illustrated in [Figure 3-16](#), the *ContributingSourcesType* class exhibits a recursive, tree-like relationship between contributing sources (it is not simply a flat relationship): a contributing source instance captures its own contributing sources, along with other content such as a description and the role of the information source. Also, the *Role* property of the *InformationSourceType* class has increased relevance when characterizing a contributing source by capturing the details of how the source contributed to its parent source.

The property table given in [Table 3-25](#) corresponds to the UML diagram shown in [Figure 3-16](#).



Table 3-25. Properties of the *ContributingSourcesType* class

Name	Type	Multiplicity	Description
<b>Source</b>	InformationSourceType	0..*	The <code>Source</code> property characterizes the organization or tool that is the contributing source. Examples of details captured include identifying characteristics, time-related attributes, and a list of the tools used to collect the information.

### 3.3.4 ExploitTargetsType Class

The `ExploitTargetsType` class specifies a set of zero or more Exploit Targets<sup>4</sup>.

The property table of the `ExploitTargetsType` class is given in [Table 3-26](#).

Table 3-26. Properties of the *ExploitTargetsType* class

Name	Type	Multiplicity	Description
<b>Exploit_Target</b>	ExploitTargetBaseType	0..*	The <code>Exploit_Target</code> property characterizes an Exploit Target to be considered with respect to one or more cyber threats. The <code>ExploitTargetBaseType</code> class is a minimal base class that is intended to be extended. The default and strongly RECOMMENDED class to fully implement an Exploit Target is the <code>et:ExploitTargetType</code> class defined in <a href="#">STIX Version 1.2.1 Part 10: Exploit Target</a> . Like the base classes defined in STIX Core data model, this class is used to minimize interdependence between STIX components, not to enable or encourage conflicting syntactic variation. Through the use of the <code>idref</code> property, a reference to an Exploit Target defined elsewhere can be specified via the direct use of the <code>ExploitTargetBaseType</code> class.

### 3.3.5 NamesType Class

The `NamesType` class specifies a set of one or more names. Note that a similar class is defined in the Campaign data model (`campaign:NamesType`), which will likely be removed in the next major version of STIX; Campaign names will be then defined exclusively via this STIX Common `NamesType` class. For details on the use of the `NamesType` class, see [Section 3.2.3.1](#).

The property table of the `NamesType` class is given in [Table 3-27](#).

Table 3-27. Properties of the *NamesType* class

Name	Type	Multiplicity	Description
<b>Name</b>	VocabularyStringType	1..*	The <b>Name</b> property specifies a name used to identify a construct, such as an alias name. 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 <i>ControlledVocabularyStringType</i> class. No default vocabulary is defined in STIX v1.2.1.

### 3.3.6 ProfilesType Class

The *ProfilesType* class specifies a list of one or more STIX Profiles (a STIX Profiles document will be available in the future).

The property table of the *ProfilesType* class is given in [Table 3-28](#).

Table 3-28. Properties of the *ProfilesType* class

Name	Type	Multiplicity	Description
<b>Profile</b>	basicDataTypes:URI	1..*	The <b>Profile</b> property specifies a reference to one STIX profile using a Uniform Resource Identifier (URI).

### 3.3.7 ReferencesType Class

The *ReferencesType* class specifies a set of one or more references.

The property table of the *ReferencesType* class is given in [Table 3-29](#).

Table 3-29. Properties of the *ReferencesType* class

Name	Type	Multiplicity	Description
<b>Reference</b>	basicDataTypes:URI	1..*	The <b>Reference</b> property specifies a reference associated with the data entry using a Uniform Resource Identifier (URI).

### 3.3.8 RelatedIdentitiesType Class

The `RelatedIdentitiesType` class specifies a set of one or more identities.

The property table of the `RelatedIdentitiesType` class is given in [Table 3-30](#).

*Table 3-30. Properties of the `RelatedIdentitiesType` class*

Name	Type	Multiplicity	Description
<b>Related_Identity</b>	<code>RelatedIdentityType</code>	1..*	The <code>Related_Identity</code> property specifies a related identity associated with the data entry using a Uniform Resource Identifier (URI).

### 3.3.9 RelatedPackageRefsType Class

The `RelatedPackageRefsType` class specifies a set of zero or more references to a related STIX Package.

The property table of the `RelatedPackageRefsType` class is given in [Table 3-31](#).

*Table 3-31. Properties of the `RelatedPackageRefsType` class*

Name	Type	Multiplicity	Description
<b>Package_Reference</b>	<code>RelatedPackageRefType</code>	0..*	The <code>Package_Reference</code> property characterizes a relationship to a related STIX Package defined elsewhere.

## 3.4 Kill Chains

A cyber kill chain is a phase-based model to describe the stages of an attack. Kill chain-related classes are defined below.

### 3.4.1 KillChainsType Class

The `KillChainsType` class specifies a set of one or more specific kill chain definitions.

The property table of the `KillChainsType` class is given in [Table 3-32](#).

Table 3-32. Properties of the *KillChainsType* class

Name	Type	Multiplicity	Description
<b>Kill_Chain</b>	KillChainType	1..*	A cyber kill chain is a phase-based model to describe the stages of an attack. The <code>Kill_Chain</code> property characterizes a single kill chain that may be referenced elsewhere; for example, from within a TTP or an Indicator component.

### 3.4.1.1 KillChainType Class

A cyber kill chain is a phase-based model to describe the stages of an attack. The `KillChainType` class characterizes a kill chain definition that may be referenced elsewhere; for example, from within a TTP or an Indicator component. Information captured includes a set of one or more kill chain phases that compose the kill chain.

The property table of the `KillChainType` class is given in [Table 3-33](#).

Table 3-33. Properties of the *KillChainType* class

Name	Type	Multiplicity	Description
<b>id</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>id</code> property specifies a globally unique identifier for the kill chain definition.
<b>name</b>	<code>basicDataTypes:</code> <code>NoEmbeddedQuoteString</code>	0..1	The <code>name</code> property captures a simple name for the kill chain definition.
<b>definer</b>	<code>basicDataTypes:</code> <code>NoEmbeddedQuoteString</code>	0..1	The <code>definer</code> property specifies the organization or individual responsible for the kill chain definition.
<b>reference</b>	<code>basicDataTypes:URI</code>	0..1	The <code>reference</code> property specifies a reference associated with the kill chain using a Uniform Resource Identifier (URI).
<b>number_of_phases</b>	<code>basicDataTypes:Integer</code>	0..1	The <code>number_of_phases</code> property specifies the number of phases in the kill chain.
<b>Kill_Chain_Phase</b>	KillChainPhaseType	1..*	The <code>Kill_Chain_Phase</code> property characterizes an individual phase within the kill chain.

### 3.4.1.1.1 KillChainPhaseType Class

A cyber kill chain is a phase-based model to describe the stages of an attack, and a cyber kill chain phase is an individual phase within a kill chain definition. The `Kill_Chain_Phase` property characterizes an individual phase within a kill chain.

The property table of the `KillChainPhaseType` class is given in [Table 3-34](#).

Table 3-34. Properties of the `KillChainPhaseType` class

Name	Type	Multiplicity	Description
<b>phase_id</b>	<code>basicDataTypes:</code> <code>QualifiedName</code>	0..1	The <code>phase_id</code> property specifies a globally unique identifier for the kill chain phase. When used within a kill chain definition (defined by the <code>KillChainType</code> class), it identifies the kill chain phase being defined. When used within a kill chain reference (defined by the <code>KillChainPhaseReferenceType</code> class), it must reference an existing kill chain <code>phase_id</code> property (note that this use is similar to the use of the <code>idref</code> property elsewhere in STIX.)
<b>name</b>	<code>basicDataTypes:</code> <code>NoEmbeddedQuoteString</code>	0..1	The <code>name</code> property captures a descriptive name of the kill chain phase. If the <code>KillChainPhaseType</code> class is extended by the <code>KillChainPhaseReferenceType</code> class, this attribute SHOULD be omitted, or if it is present, it MUST match the kill chain phase name used in the referenced kill chain.
<b>ordinality</b>	<code>basicDataTypes:</code> <code>Integer</code>	0..1	The <code>ordinality</code> property specifies the ordinality (e.g., '1', '2', or '3') of this phase within the kill chain definition. It should be omitted when the associated kill chain is being referenced (defined by the <code>KillChainPhaseReferenceType</code> class), but if the property is present, it MUST match the ordinality of the corresponding phase in the kill chain that is referenced.

### 3.4.2 KillChainPhasesReferenceType Class

A cyber kill chain is a phase-based model to describe the stages of an attack, and a cyber kill chain phase is an individual phase within a kill chain definition. The `KillChainPhasesReferenceType` class specifies a set of one or more kill chain phases.

The property table of the `KillChainPhasesReferenceType` class is given in [Table 3-35](#).

Table 3-35. Properties of the *KillChainPhasesReferenceType* class

Name	Type	Multiplicity	Description
<b>Kill_Chain_Phase</b>	KillChainPhaseReferenceType	1..*	The <code>Kill_Chain_Phase</code> property specifies a single kill chain phase associated with this item.

### 3.4.2.1 KillChainPhaseReferenceType Class

The `KillChainPhaseReferenceType` class specifies a phase within a kill chain by extending the `KillChainPhaseType` superclass. A kill chain reference in an Indicator component indicates that the indicator detects malicious behavior at that phase of the kill chain. A kill chain reference in a TTP component indicates that the TTP is used (malware, infrastructure, etc.) at that phase of the kill chain.

The property table of the `KillChainPhaseReferenceType` class is given in [Table 3-36](#).

Table 3-36. Properties of the *KillChainPhaseReferenceType* class

Name	Type	Multiplicity	Description
<b>kill_chain_id</b>	basicDataTypes: QualifiedName	0..1	The <code>kill_chain_id</code> property specifies the globally unique identifier for the referenced kill chain instance.
<b>kill_chain_name</b>	basicDataTypes: NoEmbeddedQuotesString	0..1	The <code>kill_chain_name</code> property captures a descriptive name of the kill chain. If a kill chain is referenced by the <code>kill_chain_id</code> property, this attribute <b>SHOULD</b> be omitted, or if it is present, it <b>MUST</b> match the <code>name</code> property of the corresponding kill chain phase that is referenced.

## 3.5 General Shared Classes

Unlike the classes defined in the previous sections that shared similar roles, the following classes serve a variety of purposes and are shared by the collection of STIX data models.

### 3.5.1 ActivityType Class

The `ActivityType` class characterizes basic information about an activity a defender might use. It is an abstract class, so it **MUST** be extended via a subclass to express additional activity information. STIX does not define a default subclass. Note that an activity defined by the `ActivityType` class is fairly simple and includes only date/time information and a description. By contrast, a Course of Action construct contains detailed information such as the

stage in the cyber threat management lifecycle to which the course of action is relevant, the impact and cost of applying the course of action, and efficacy of the course of action.

The property table for the `ActivityType` class is given in [Table 3-37](#).

Table 3-37. Properties of the `ActivityType` class

Name	Type	Multiplicity	Description
Date_Time	DateTimeWithPrecisionType	1	The <code>Date_Time</code> property specifies the date and time at which the activity occurred. To avoid ambiguity, all timestamps SHOULD include a specification of the time zone. In addition to specifying a date and time, the <code>Date_Time</code> property may also capture a <code>precision</code> property to specify the granularity with which the time should be considered, as specified by the <code>DateTimePrecisionEnum</code> 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.
Description	StructuredTextType	0..*	The <code>Description</code> property captures a textual description of the activity. Any length is permitted. Optional formatting is supported via the <code>structuring_format</code> property of the <code>StructuredTextType</code> class.

### 3.5.2 AddressAbstractType Class

The `AddressAbstractType` class characterizes geographic address information. It is an abstract class, so it MUST be extended via a subclass to capture an address. STIX v1.2.1 defines the `CIQAddress3.0InstanceType` subclass as a default extension to leverage the OASIS Customer Information Quality (CIQ) data model, which is a set of XML specifications for representing characteristic information about individuals and organizations. Details are provided in [STIX Version 1.2.1 Part 12: Default Extensions](#).

The `AddressAbstractType` class has no properties of its own (so there is no associated property table). Its UML diagram is illustrated in [Figure 3-17](#).

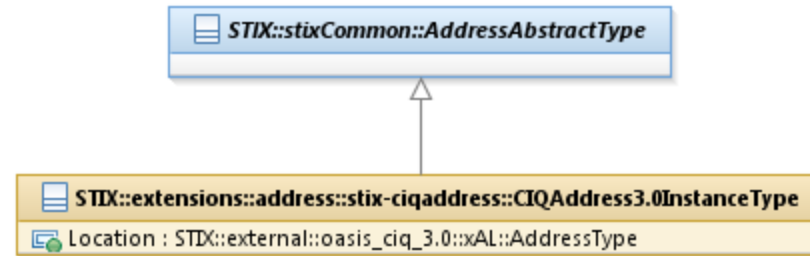


Figure 3-17. UML diagram of the *AddressAbstractType* class

### 3.5.3 ConfidenceType Class

The *ConfidenceType* class characterizes confidence in an assertion.

The property table of the *ConfidenceType* class is given in [Table 3-38](#).

Table 3-38. Properties of the *ConfidenceType* class

Name	Type	Multiplicity	Description
<b>timestamp</b>	<code>basicDataTypes:DateTime</code>	0..1	The <code>timestamp</code> property specifies the date and time of the confidence assertion. To avoid ambiguity, all timestamps SHOULD include a specification of the time zone.
<b>timestamp_precision</b>	<code>DateTimePrecisionEnum</code>	0..1	The <code>timestamp_precision</code> property specifies the granularity with which the <code>timestamp</code> property should be considered, as specified by the <code>DateTimePrecisionEnum</code> 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.
<b>Value</b>	<code>VocabularyStringType</code>	0..1	The <code>Value</code> property specifies the level of confidence held in this direct assertion. Examples of potential levels include <i>high</i> , <i>medium</i> , and <i>low</i> (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 <code>ControlledVocabularyStringType</code> class. The STIX default vocabulary class for use in the property is <i>'HighMediumLowVocab-1.0.'</i>
<b>Description</b>	<code>StructuredTextType</code>	0..*	The <code>Description</code> property captures a textual description of the confidence assertion. Any length is permitted. Optional formatting is supported via the <code>structuring_format</code> property of the <code>StructuredTextType</code> class.
<b>Source</b>	<code>InformationSourceType</code>	0..1	The <code>Source</code> property characterizes the organization or tool that is the source of the confidence assertion. Examples of details captured include identifying characteristics, time-related attributes, and a list of the tools used to collect the information.
<b>Confidence_Assertion_Chain</b>	<code>ConfidenceAssertionChainType</code>	0..1	The <code>Confidence_Assertion_Chain</code> property specifies a set of one or more related confidence assertions.

### 3.5.4 IdentityType Class

The `IdentityType` class characterizes identity information for both individuals and organizations. The `IdentityType` class is intended to be extended via a subclass to capture the structured descriptions of identity information for both individuals and organizations. STIX v1.2.1 defines a default extension to the `IdentityType` class to leverage the OASIS Customer Information Quality (CIQ) data model, which is a set of XML specifications for representing characteristic information about individuals and organizations (see [STIX Version 1.2.1 Part 12: Default Extensions](#)).

The property table of the `IdentityType` class is given in [Table 3-39](#).

Table 3-39. Properties of the *IdentityType* class

Name	Type	Multiplicity	Description
<b>id</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>id</code> property specifies a globally unique identifier for the identity.
<b>idref</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>idref</code> property specifies a globally unique identifier for an identity specified elsewhere. When the <code>idref</code> property is used, the <code>id</code> property MUST NOT also be specified and the other properties of the <i>IdentityType</i> class SHOULD NOT hold any content.
<b>Name</b>	<code>basicDataTypes:BasicString</code>	0..1	The <code>Name</code> property captures a simple name for the identity.
<b>Related_Identities</b>	<code>RelatedIdentitiesType</code>	0..1	The <code>Related_Identities</code> property specifies a set of one or more identities related to this identify.

### 3.5.5 InformationSourceType Class

The *InformationSourceType* class specifies source information for the given STIX content.

The property table of the *InformationSourceType* class is given in [Table 3-40](#).

Table 3-40. Properties of the *InformationSourceType* class

Name	Type	Multiplicity	Description
<b>Description</b>	<code>StructuredTextType</code>	0..*	The <code>Description</code> property captures a textual description of the information source. Any length is permitted. Optional formatting is supported via the <code>structuring_format</code> property of the <i>StructuredTextType</i> class.
<b>Identity</b>	<code>IdentityType</code>	0..1	The <code>Identity</code> property characterizes the identity of the information source. For situations calling for more than a simple name, the underlying class may be extended using a more complete structure such as the <code>CIQIdentity3.0InstanceType</code> subclass as defined in <a href="#">STIX Version 1.2.1 Part 12: Default Extensions</a> .

<b>Role</b>	VocabularyStringType	0..*	The <code>Role</code> property specifies a role played by the information source. Examples of potential roles include <i>initial author</i> , <i>aggregator</i> , and <i>transformer/translator</i> (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 <code>ControlledVocabularyStringType</code> class. The STIX default vocabulary class for use in the property is ' <i>InformationSourceRoleVocab-1.0</i> '.
<b>Contributing_Sources</b>	ContributingSourcesType	0..1	The <code>Contributing_Sources</code> property specifies a set of zero or more individual sources contributing to the STIX content. Note that this property captures <i>secondary</i> sources of information.
<b>Time</b>	TimeType	0..1	The <code>Time</code> property characterizes the time-related attributes of the STIX content.
<b>Tools</b>	cybox: ToolsInformationType	0..1	The <code>Tools</code> property specifies a set of one or more tools contributing to the STIX content.
<b>References</b>	ReferencesType	0..1	The <code>References</code> property specifies a set of one or more references to information source material related to the STIX content.

### 3.5.6 StatementType Class

The `StatementType` class characterizes a statement associated with STIX content.

The UML diagram corresponding to the `StatementType` class is shown in [Figure 3-18](#).



Figure 3-18. UML diagram of the *StatementType* class

The property table given in [Table 3-41](#) corresponds to the UML diagram shown in [Figure 3-18](#).

Table 3-41. Properties of the *StatementType* class

Name	Type	Multiplicity	Description
<b>timestamp</b>	<code>basicDataTypes:DateTime</code>	0..1	The <code>timestamp</code> property specifies the date and time of the statement. To avoid ambiguity, all timestamps SHOULD include a specification of the time zone.
<b>timestamp_precision</b>	<code>DateTimePrecisionEnum</code>	0..1	The <code>timestamp_precision</code> property specifies the granularity with which the <code>timestamp</code> property should be considered, as specified by the <code>DateTimePrecisionEnum</code> 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.
<b>Value</b>	<code>VocabularyStringType</code>	0..1	The <code>Value</code> property specifies the statement's level of importance. Examples of potential levels include <i>high</i> , <i>medium</i> , and <i>low</i> (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 <code>ControlledVocabularyStringType</code> class. The STIX default vocabulary class for use in the property is 'HighMediumLowVocab-1.0'; however, when the <code>StatementType</code> class is used, this default

			vocabulary is often overwritten by a different vocabulary more suitable to the context.
<b>Description</b>	StructuredTextType	0..*	The <code>Description</code> property captures a textual description of the statement. Any length is permitted. Optional formatting is supported via the <code>structuring_format</code> property of the <code>StructuredTextType</code> class.
<b>Source</b>	InformationSourceType	0..1	The <code>Source</code> property characterizes the organization or tool that is the source of the statement. Examples of details captured include identifying characteristics, time-related attributes, and a list of the tools used to collect the information.
<b>Confidence</b>	ConfidenceType	0..1	The <code>Confidence</code> property characterizes the confidence in the statement.

### 3.5.7 ToolInformationType Class

The `ToolInformationType` class characterizes information about a hardware or software tool by extending the CybOX `ToolInformationType` superclass (note that both classes are named “`ToolInformationType`” but are in different UML packages).

The UML diagram corresponding to the `ToolInformationType` class is shown in [Figure 3-19](#).

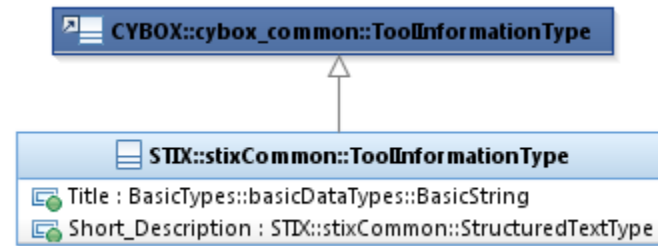


Figure 3-19. UML diagram of the `ToolInformationType` class

The property table given in [Table 3-42](#) corresponds to the UML diagram shown in [Figure 3-19](#).

Table 3-42. Properties of the *ToolInformationType* class

Name	Type	Multiplicity	Description
<b>Title</b>	<code>basicDataTypes:BasicString</code>	0..1	The <code>Title</code> property provides a simple title for the tool and reflects what the producer thinks the tool as a whole should be called. Titles are typically used by humans to reference a particular tool; however, titles are not meant to be used for correlation.
<b>Short_Description</b>	<code>StructuredTextType</code>	0..*	The <code>Short_Description</code> property captures a short textual description of the tool.

## 3.6 General Data Types

### 3.6.1 DateTimeWithPrecisionType Data Type

The `DateTimeWithPrecisionType` data type specializes the `basicDataTypes:DateTime` data type by capturing precision information.

The property table of the `DateTimeWithPrecisionType` data type is provided in [Table 3-43](#).

Table 3-43. Properties of the *DateTimeWithPrecisionType* class

Name	Type	Multiplicity	Description
<b>precision</b>	<code>DateTimePrecisionEnum</code>	0..1	The <code>precision</code> property specifies the granularity with which a timestamp should be considered as specified by the <code>DateTimePrecisionEnum</code> enumeration (e.g., <i>'hour'</i> , <i>'minute'</i> ). If omitted, the default precision is <i>'second'</i> . Digits in a timestamp that are beyond the specified precision <b>SHOULD</b> be zeroed out.

Some classes, such as the `ConfidenceType` and `StatementType` classes, have both a “timestamp” property (defined by the `basicDataTypes:DateTime` data type) and a “timestamp precision” property (defined by the `DateTimePrecisionEnum` enumeration). This is due to the evolution of the STIX language; a future major release could include the replacement of these two properties with a single property defined by the `DateTimeWithPrecisionType` class.

### 3.6.2 NativeFormatStringType Data Type

The `NativeFormatStringType` data type specializes the `basicDataTypes:BasicString` data type in order to capture data in the native format of some external language (see [STIX Version 1.2.1 Part 12: Default Extensions](#)). The data may be encoded in Base64 per [\[RFC4648\]](#). Data encoded in Base64 must be denoted as such using the `encoded` property.

The property table of the `NativeFormatStringType` data type is provided in [Table 3-44](#).

Table 3-44. Properties of the `NativeFormatStringType` class

Name	Type	Multiplicity	Description
<b>encoded</b>	<code>basicDataTypes:Boolean</code>	0..1	The <code>encoded</code> property specifies whether the data is Base64 encoded (' <i>true</i> ') or not encoded (' <i>false</i> '). The default value is ' <i>false</i> '.

### 3.6.3 StructuredTextType Data Type

The `StructuredTextType` data type specializes the `basicDataTypes:BasicString` data type by capturing structuring format information. An identifier is captured to allow for differential data marking of distinct construct instances, and an ordinality property indicates the order of the instances if more than one is specified.

The property table of the `StructuredTextType` data type is provided in [Table 3-45](#).

Table 3-45. Properties of the `StructuredTextType` data type

Name	Type	Multiplicity	Description
<b>id</b>	<code>basicDataTypes:QualifiedName</code>	0..1	The <code>id</code> property specifies a globally unique identifier for the construct instance.
<b>ordinality</b>	<code>basicDataTypes:PositiveInteger</code>	0..1	The <code>ordinality</code> property specifies the order (e.g., '1', '2', or '3') of this construct instance (e.g., Description) within a potential set of multiple peer construct instances. If only a single construct instance is present, its ordinality can be assumed to be 1. If multiple construct instances are present, the <code>ordinality</code> property SHOULD be specified with unique values for each instance.

<b>structuring_format</b>	<code>basicDataTypes:BasicString</code>	0..1	The <code>structuring_format</code> property specifies the structuring format used within an instance of the <code>StructuredTextType</code> class. If this property is absent, then markup <b>MUST NOT</b> be used.
---------------------------	---	------	--

### 3.7 Vocabulary Data Types

There are three vocabulary-related UML data types defined in the Common data model, and together they provide a content creator with four choices for defining content, listed below in order of formality. Please see [STIX Version 1.2.1 Part 14: Vocabularies](#) for further information on STIX vocabularies.

- Leverage a default vocabulary using the `ControlledVocabularyStringType` data type. STIX v1.2.1 defines a collection of default vocabularies and associated enumerations that are based on input from the STIX community (see [STIX Version 1.2.1 Part 14: Vocabularies](#)); however, not all vocabulary properties have an assigned default vocabulary.
- Formally define a custom vocabulary using the `ControlledVocabularyStringType` data type. To achieve value enforcement, a custom vocabulary must be formally added to the STIX Vocabulary data model. Because this is an extension of the STIX Vocabulary data model, producers and consumers **MUST** be aware of the addition to the data model for successful sharing of STIX documents.
- Reference an externally-defined, custom vocabulary using the `UnenforcedVocabularyStringType` data type to constrain the set of values. Externally-defined vocabularies are publically defined, but have not been included as formally specified vocabularies within the STIX Vocabulary data model using the `ControlledVocabularyStringType` data type. In this case, it is sufficient to specify the name of the vocabulary and a URL that defines that vocabulary.
- Choose an arbitrary and unconstrained value using the `VocabularyStringType` data type.

While not required by the general STIX language, default vocabularies should be used whenever possible to ensure the greatest level of compatibility between STIX users. If an appropriate default vocabulary is not available a formally defined custom vocabulary can be specified and leveraged. In addition to compatibility advantages, using formally defined vocabularies (whether default vocabularies or otherwise defined) enables enforced use of valid enumeration values; please see [STIX Version 1.2.1 Part 14: Vocabularies](#) for the associated policy.

If a formally defined vocabulary is not sufficient for a content producer's purposes, the STIX Vocabulary data model allows the two alternatives listed above: externally defined custom vocabularies and arbitrary string values, which dispense with enumerated vocabularies altogether. If a custom vocabulary is not formally added to the Vocabulary data model then no enforcement policy of appropriate values is specified.

The UML diagram shown in [Figure 3-20](#) illustrates the relationships between the three vocabulary data types defined in the STIX Common data model. As illustrated, all controlled vocabularies formally defined within the STIX Vocabulary data model are defined using an enumeration derived from the `ControlledVocabularyStringType` data type.



As shown, the `HighMediumLowVocab-1.0` enumeration (used as a defined controlled vocabulary exemplar) is defined as a specialization of the `ControlledVocabularyStringType` data type, and therefore it is also a specialization of the `VocabularyStringType` data type.

Further details of each vocabulary class are provided in Subsections 3.7.1 through 3.7.3.

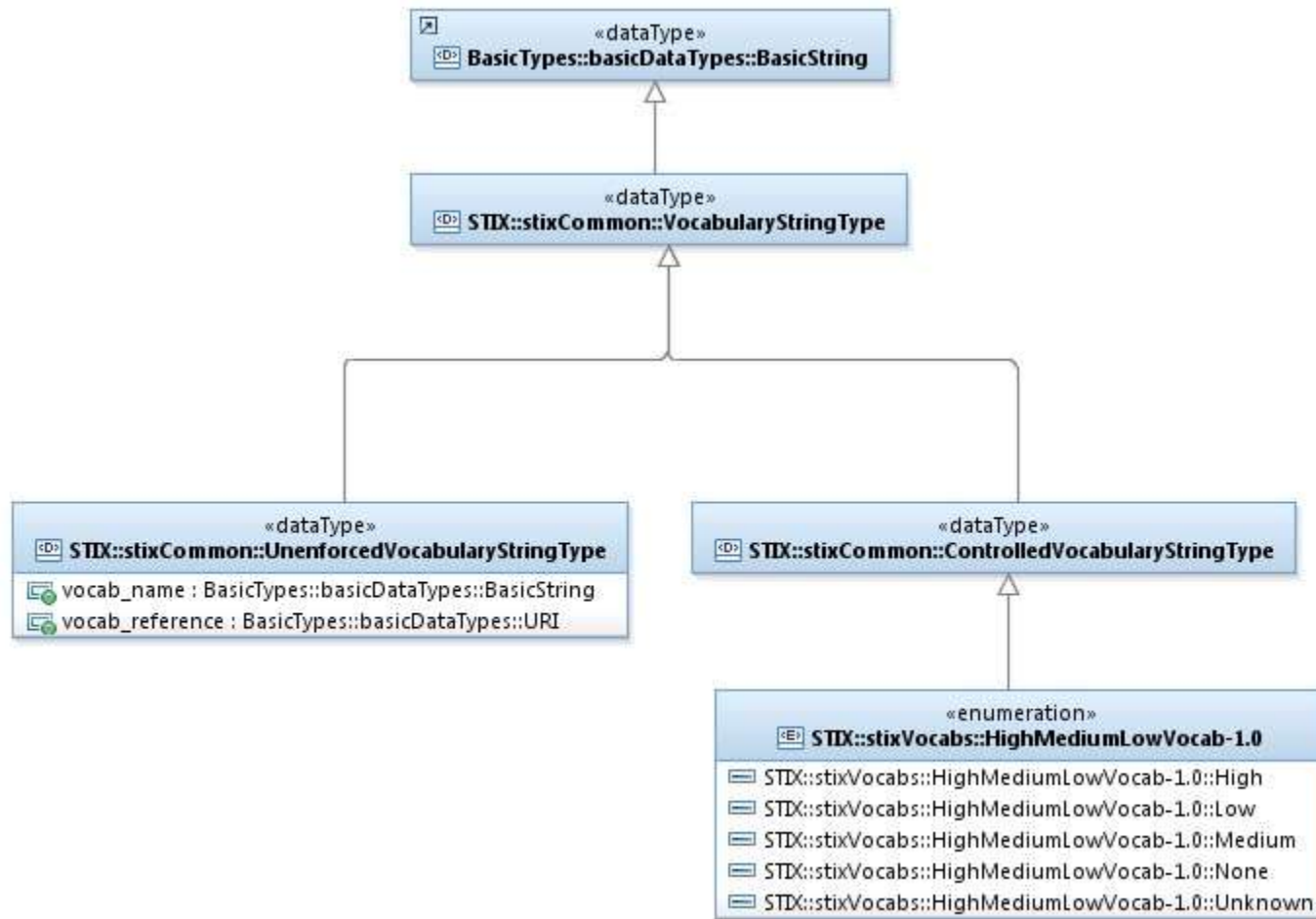


Figure 3-20. UML diagram of the STIX<sup>TM</sup> Vocabulary data model

### 3.7.1 VocabularyStringType Data Type

The `VocabularyStringType` data type is the basic data type of all vocabularies. Therefore, all properties in the collection of STIX data models that makes use of the Vocabulary data model must be defined to use the `VocabularyStringType` data type. Because this data type is a specialization of the `basicDataTypes:BasicString` data type, it can be used to support the arbitrary string option for vocabularies.

### 3.7.2 UnenforcedVocabularyStringType Data Type

The `UnenforcedVocabularyStringType` data type specifies custom vocabulary values via an enumeration defined outside of the STIX Vocabulary data model. It extends the `VocabularyStringType` data type. Note that the STIX vocabulary data model does not define any enforcement policy for this data type.

The property table of the `UnenforcedVocabularyStringType` data type is given in [Table 3-46](#).

Table 3-46. Properties of the `UnenforcedVocabularyStringType` data type

Name	Type	Multiplicity	Description
<b>vocab_name</b>	<code>basicDataTypes:</code> <code>NoEmbeddedQuoteString</code>	0..1	The <code>vocab_name</code> property specifies the name of the externally defined vocabulary.
<b>vocab_reference</b>	<code>basicDataTypes:URI</code>	0..1	The <code>vocab_reference</code> property specifies the location of the externally defined vocabulary using a Uniform Resource Identifier (URI).

### 3.7.3 ControlledVocabularyStringType Data Type

The `ControlledVocabularyStringType` data type specifies a formally defined vocabulary. It is an abstract data type so it MUST be extended via an enumeration from the STIX Vocabulary data model (descriptions of all default vocabularies defined within the STIX Vocabulary data model are found in [STIX Version 1.2.1 Part 14: Vocabularies<sup>5</sup>](#)). Any custom vocabulary must be defined via an enumeration added to the STIX Vocabulary data model, if appropriate enumeration values are to be enforced.

The `ControlledVocabularyStringType` class has no properties of its own, so there is no associated property table.

## 3.8 Enumerations

### 3.8.1 DateTimePrecisionEnum Enumeration

The `DateTimePrecisionEnum` enumeration is an inventory of values for representing time precision. The enumeration literals are given in [Table 3-47](#).

*Table 3-47. Literals of the `DateTimePrecisionEnum` enumeration*

Enumeration Literals	Description
<b>year</b>	The given date/time is precise to the given year.
<b>month</b>	The given date/time is precise to the given month.
<b>day</b>	The given date/time is precise to the given day.
<b>hour</b>	The given date/time is precise to the given hour.
<b>minute</b>	The given date/time is precise to the given minute.
<b>second</b>	The given date/time is precise to the given second (including fractional seconds).

### 3.8.2 RelationshipScopeEnum

The `RelationshipScopeEnum` enumeration is an inventory of types of relationship groupings between a subject and a collection of objects. The enumeration literals are given in [Table 3-48](#).

*Table 3-48. Literals of the `RelationshipScopeEnum` enumeration*

Enumeration Literals	Description
<b>inclusive</b>	A single relationship is being defined between the subject and the collection of objects indicated by the related items.
<b>exclusive</b>	Multiple relationships are being defined between the subject and each object individually.



---

## 4 Conformance

Implementations have discretion over which parts (components, properties, extensions, controlled vocabularies, etc.) of STIX they implement (e.g., Indicator/Suggested\_COAs).

[1] Conformant implementations must conform to all normative structural specifications of the UML model or additional normative statements within this document that apply to the portions of STIX they implement (e.g., Implementers of the entire TTP component must conform to all normative structural specifications of the UML model or additional normative statements within this document regarding the TTP component).

[2] Conformant implementations are free to ignore normative structural specifications of the UML model or additional normative statements within this document that do not apply to the portions of STIX they implement (e.g., Non-implementers of any particular properties of the TTP component are free to ignore all normative structural specifications of the UML model or additional normative statements within this document regarding those properties of the TTP component).

The conformance section of this document is intentionally broad and attempts to reiterate what already exists in this document. The STIX 1.2 Specifications, which this specification is based on, did not have a conformance section. Instead, the STIX 1.2 Specifications relied on normative statements and the non-mandatory implementation of STIX profiles. STIX 1.2.1 represents a minimal change from STIX 1.2, and in that spirit no requirements have been added, modified, or removed by this section.

---

## Appendix A. Acknowledgments

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

### Participants:

Dean Thompson, Australia and New Zealand Banking Group (ANZ Bank)  
Bret Jordan, Blue Coat Systems, Inc.  
Adnan Baykal, Center for Internet Security (CIS)  
Jyoti Verma, Cisco Systems  
Liron Schiff, Comilion (mobile) Ltd.  
Jane Ginn, Cyber Threat Intelligence Network, Inc. (CTIN)  
Richard Struse, DHS Office of Cybersecurity and Communications (CS&C)  
Marlon Taylor, DHS Office of Cybersecurity and Communications (CS&C)  
David Eilken, Financial Services Information Sharing and Analysis Center (FS-ISAC)  
Sarah Brown, Fox-IT  
Ryusuke Masuoka, Fujitsu Limited  
Eric Burger, Georgetown University  
Jason Keirstead, IBM  
Paul Martini, iboss, Inc.  
Jerome Athias, Individual  
Terry MacDonald, Individual  
Alex Pinto, Individual  
Patrick Maroney, Integrated Networking Technologies, Inc.  
Wouter Bolsterlee, Intelworks BV  
Joep Gommers, Intelworks BV  
Sergey Polzunov, Intelworks BV  
Rutger Prins, Intelworks BV  
Andrei Sirghi, Intelworks BV  
Raymon van der Velde, Intelworks BV  
Jonathan Baker, MITRE Corporation  
Sean Barnum, MITRE Corporation  
Desiree Beck, MITRE Corporation  
Mark Davidson, MITRE Corporation  
Ivan Kirillov, MITRE Corporation  
Jon Salwen, MITRE Corporation  
John Wunder, MITRE Corporation  
Mike Boyle, National Security Agency  
Jessica Fitzgerald-McKay, National Security Agency  
Takahiro Kakumaru, NEC Corporation  
John-Mark Gurney, New Context Services, Inc.  
Christian Hunt, New Context Services, Inc.  
Daniel Riedel, New Context Services, Inc.  
Andrew Storms, New Context Services, Inc.  
John Tolbert, Queralt, Inc.  
Igor Baikalov, Securonix  
Bernd Grobauer, Siemens AG  
Jonathan Bush, Soltra  
Aharon Chernin, Soltra  
Trey Darley, Soltra  
Paul Dion, Soltra  
Ali Khan, Soltra  
Natalie Suarez, Soltra  
Cedric LeRoux, Splunk Inc.  
Brian Luger, Splunk Inc.

Crystal Hayes, The Boeing Company  
Brad Butts, U.S. Bank  
Mona Magathan, U.S. Bank  
Adam Cooper, United Kingdom Cabinet Office  
Mike McLellan, United Kingdom Cabinet Office  
Chris O'Brien, United Kingdom Cabinet Office  
Julian White, United Kingdom Cabinet Office  
Anthony Rutkowski, Yaana Technologies, LLC

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

---

## Appendix B. Revision History

Revision	Date	Editor	Changes Made
wd01	21 August 2015	Sean Barnum Desiree Beck Aharon Chernin Rich Piazza	Initial transfer to OASIS template

Notes \_\_\_\_\_

<sup>1</sup> The CybOX™ Observable data model is actually defined in the [CybOX Language](#), not in STIX; but it is included in the list because it is referenced often from STIX.

<sup>2</sup> There is currently no base class defined for the Observable component, which is defined in the [CybOX Language](#).

<sup>3</sup> There is currently no base class defined for the Observable component, which is defined in the [CybOX Language](#).

<sup>4</sup> This class will eventually be moved to the STIX Core data model so that its location is consistent with similar classes (e.g., `IncidentsType`, `CoursesOfActionType`). The move will require a major version change because while instance content will not change, STIX bindings and APIs will need to import the class from another namespace and therefore may break.

<sup>5</sup> Note that all defined vocabulary enumerations have version numbers in their names to facilitate additions to the enumerations that are backward compatible.