STIX 2.0 Specification

Core Concepts

Version 2.0-draft-1

Document Table of Contents

1. Technical Committee

OASIS Cyber Threat Intelligence (CTI) TC

2. Chair

Richard Struse (Richard.Struse@hq.dhs.gov), DHS Office of Cybersecurity and Communications (CS&C)

3. Editors

Bret Jordan (bret.jordan@bluecoat.com), Blue Coat Systems, Inc.
John Wunder (jwunder@mitre.org), MITRE Corporation

4. Acknowledgments

STIX Subcommittee Chairs:
John Wunder (jwunder@mitre.org), MITRE Corporation
Aharon Chernin (achernin@soltra.com), Soltra

Special Thanks:
The following individuals made substantial contributions to this specification in the form of normative text and proofing and their contributions are gratefully acknowledged:

- Bret Jordan, Blue Coat Systems, Inc.
- Terry MacDonald, Cosive
- Jane Ginn, Cyber Threat Intelligence Network, Inc. (CTIN)
- Jason Keirstead, IBM
- Tim Casey, Intel
- Allan Thomson, LookingGlass Cyber
- John Wunder, MITRE Corporation
- Richard Piazza, MITRE Corporation
- John-Mark Gurney, New Context Services, Inc.
- Iain Brown, United Kingdom Cabinet Office

Contributors:
The following individuals were members of the OASIS CTI STIX Technical Committee during the creation of this specification and their contributions are gratefully acknowledged:
<todo, make sure this is up to date>

- Dean Thompson, Australia and New Zealand Bank
- Alexander Foley, Bank of America
- Bret Jordan, Blue Coat Systems, Inc.
- Sarah Kelley, Center for Internet Security (CIS)
- Alexandre Dulaunoy, CIRCL
- Ted Bedwell, Cisco Systems
- Craig Brozefsky, Cisco Systems
- Jyoti Verma, Cisco Systems
- Jane Ginn, Cyber Threat Intelligence Network, Inc. (CTIN)
- Richard Struse, DHS Office of Cybersecurity and Communications
- Marlon Taylor, DHS Office of Cybersecurity and Communications (CS&C)
- Gordan Hundley, DTCC
- Wouter Bolsterlee, EclecticIQ
- Joep Gommers, EclecticIQ
- Sergey Rolzunov, EclecticIQ
- Rutger Prins, EclecticIQ
- Andrei Sirghi, EclecticIQ
- Raymon van der Velde, EclecticIQ
- Ravi Sharda, EMC
- David Eilken, FS-ISAC
- Paul Patrick, FireEye, Inc.
- Sarah Brown, Fox-IT
- Ryusuke Masuoka, Fujitsu Limited
- Daisuke Murabayashi, Fujitsu Limited
- Eric Burger, Georgetown University
- Masato Terada, Hitachi, Ltd.
- Jason Keirstead, IBM
- Paul Martini, iboss, Inc.
- Jerome Athias, Individual
- Sanjiv Kalkar, Individual
- Terry MacDonald, Cosive
- Alex Pinto, Individual
- Tim Casey, Intel Corporation
- Julie Modlin, Johns Hopkins University, Applied Physics Laboratory
- Mark Moss, Johns Hopkins University, Applied Physics Laboratory
- Pamela Smith, Johns Hopkins University, Applied Physics Laboratory
- Allan Thomson, LookingGlass Cyber

- Greg Back, MITRE Corporation
- Jonathan Baker, MITRE Corporation
- Sean Barnum, MITRE Corporation
- Nicole Gong, MITRE Corporation
- Ivan Kirillov, MITRE Corporation
- Richard Piazza, MITRE Corporation
- Jon Salwen, MITRE Corporation
- Emmanuelle Vargas-Gonzalez, MITRE Corporation
- Bryan Worrell, 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.
- Robin Cover, OASIS
- Chet Ensign, OASIS
- Cory Casanave, Object Management Group
- John Tolbert, Queralt, Inc.
- Igor Baikalov, Securonix
- Bernd Grobauer, Siemens AG
- John Anderson, Soltra
- Aharon Chernin, Soltra
- Trey Darley, Kingfisher Operations, sprl
- Mark Davidson, Soltra
- Paul Dion, Soltra
- Daniel Dye, Soltra
- Ali Khan, Soltra
- Natalie Suarez, Soltra
- Cedric LeRoux, Splunk, Inc.
- Brian Luger, Splunk, Inc.
- Tom Blauvelt, Symantec Corp.
- Crystal Hayes, The Boeing Company
- Andrew Pendergast, ThreatConnect, Inc.
- Brad Butts, U.S. Bank
- Brian Fay, U.S. Bank
- Mona Magathan, U.S. Bank
- Yevgen Sautin, U.S. Bank
- Iain Brown, United Kingdom Cabinet Office
- Adam Cooper, United Kingdom Cabinet Office
- Mike McLellan, United Kingdom Cabinet Office

- Chris O'Brien, United Kingdom Cabinet Office
- Howard Staple, United Kingdom Cabinet Office
- Chris Taylor, United Kingdom Cabinet Office
- Laurie Thomson, United Kingdom Cabinet Office
- Julian White, United Kingdom Cabinet Office
- Bethany Yates, United Kingdom Cabinet Office
- Eoghan Casey, US Department of Defense (DoD)
- Gary Katz, US Department of Defense (DoD)
- Jeff Mates, US Department of Defense (DoD)
- Robert Coderre, VeriSign
- Haripriya Gajendran, VeriSign
- Kyle Maxwell, VeriSign
- Eric Osterweil, VeriSign
- Anthony Rutkowski, Yaana Technologies, LLC

5. Abstract

Structured Threat Information Expression (STIX) is an information model and serialization for cyber threat intelligence (CTI). By allowing the consistent expression of CTI in a machine-readable specification, STIX supports shared threat analysis, machine automation, and information sharing. It enables use cases such as indicator exchange, management of response activities, shared malware analysis, and higher-level threat intelligence sharing.

This specification document defines the domain objects, relationship objects, and vocabularies that compose STIX 2.0. In addition, it includes the names of common types used in STIX, a discussion of how STIX properties and objects can be customized, and definitions of types necessary for data handling and for transporting STIX content.

6. Introduction

Structured Threat Information Expression (STIX) is an information model and serialization for cyber threat intelligence (CTI). By allowing the consistent expression of CTI in a machine-readable specification, STIX supports shared threat analysis, machine automation, and information sharing. It enables use cases such as indicator exchange, management of response activities, shared malware analysis, and higher-level threat intelligence sharing.

In the interest of developing a coherent and consistent community-driven model, STIX has been significantly redesigned and, as a result, captures a subset of the objects and fields defined in STIX 1.2. The objects chosen for inclusion in STIX 2.0 represent a minimally viable product to fulfill basic consumer and producer requirements for cyber threat intelligence sharing. Objects

and fields not included in STIX 2.0 but deemed necessary by the community will be included in future 2.x releases (development of STIX 2.1 will immediately follow the STIX 2.0 release).

6.1. 6.1.Purpose

The purpose of the STIX information model is to specify, characterize, and capture CTI. It supports two primary activities: shared threat analysis and machine automation. STIX addresses a full range of cyber threat use cases – including threat analysis, capture and specification of indicators, management of response activities, and information sharing – to improve consistency, efficiency, interoperability, and overall situational awareness.

6.2. Overview

6.2.1. Graph-Based Model

At a high level, the STIX 2.0 information model is graph-based: STIX Domain Objects (SDOs) define the graph nodes and STIX Relationship Objects (SROs, primarily Relationship itself) define the edges. This graph-based model conforms to common analysis approaches and allows for flexible, yet structured and consistent, representations of intelligence information. For example, it allows for the sharing of an Indicator and a Relationship saying that the indicator can detect a piece of Malware. However, it then also allows for a new Relationship to be published, even by a different organization, that says that that Malware is used by a certain Threat Actor.

Every domain object and relationship object in STIX is built on a common object definition, providing capabilities such as versioning, data marking (representing how data can be shared and used), and extensibility.

6.2.1.1. Domain Objects

STIX 2.0 defines 14 STIX Domain Objects (SDOs): Attack Pattern, Campaign, Course of Action, Incident, Indicator, Intrusion Set, Malware, Observed Data, Report, Source, Threat Actor,Tool, Victim Target, and Vulnerability. Those objects each correspond to an entity commonly represented in cyber threat intelligence. Using the relationships, they can then be used as building blocks and composed into broader intelligence pictures.

6.2.1.2. Relationships

There are several different types of relationships in STIX: *factual* relationships, such as the source that created a piece of STIX content, are represented as references embedded within objects. *Intelligence* relationships, on the other hand, link together SDOs to make some analytic assertion. An Incident can be linked to a Campaign, for example, to assert that the Incident is a part of that Campaign. STIX defines 44 named intelligence relationships that are represented using the Relationship SRO. These named relationships have community consensus meaning and are defined between particular SDOs. The Relationship object can also be used to

represent relationships between any SDO and any other SDO using either a simple "related-to" named relationship or by using user-defined relationship names. Finally, other forms of SROs define special intelligence relationships with extra data. Sighting, for example, defines counts and start/end times to capture extra data on the relationship between something that was seen and the observations themselves.

6.2.2. Vocabularies and Common Types

STIX 2.0 also defines a set of standard vocabularies to further describe SDOs and relationship objects. Vocabularies in STIX are generally "open": while the STIX specification defines a set of suggested terms that are common, well-defined, and well-understood, end users are free to expand on these terms.

STIX also defines a number of common types, used throughout the SDOs and SROs, and types used to apply markings to data

6.2.3. Serialization

STIX is designed to support implementation in different serializations, but the mandatory-to-implement serialization in STIX 2.0 is JSON. This means that all STIX-compatible tools must support JSON as a serialization and can, optionally, support additional serializations.

Due to the selection of JSON as the mandatory-to-implement serialization, all examples in the document are written in JSON.

6.2.4. Sharing STIX

STIX is designed to be transport-agnostic. The structures and serializations do not require any particular mechanism to exchange STIX, and the "bundle" object was defined to allow for transportation in data-agnostic specifications. A companion CTI specification, however, called TAXII (TODO REF) was designed specifically to transport STIX Objects and is the recommended mechanism to do so.

6.3. Requirements

The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [add reference].

An implementation is not compliant if it fails to satisfy one or more of the **MUST** or **REQUIRED** level requirements. An implementation that satisfies all the **MUST** or **REQUIRED** level and all the **SHOULD** level requirements is said to be “unconditionally compliant”; one that satisfies all the **MUST** level requirements but not all the **SHOULD** level requirements is said to be “conditionally compliant”.

6.4. Document Structure

This specification document is structured as follows.
<TODO - CHECK THE ORDER OF THE SECTIONS AND ORGANISE THIS PARA AS APPROPRIATE>
Section <TODO> defines the common types used throughout STIX, and is referenced in other sections of this specification. Information on customizing STIX can be found in Section 4<TODO>, with guidance, requirements and examples of Custom Properties. Section 5<TODO> outlines how STIX Objects are transported or transmitted. Section 6<TODO> defines the properties and behaviors common to all STIX Domain Objects (SDOs).

<to do> FINISH EXPANDING THIS SECTION - Suggest we do this when we have combined all the STIX docs into one. Easier then to get order of sections correct.

6.5. Document Conventions

6.5.1. Font Colors and Style

The following color, font and font style conventions are used in this document:

- The `Consolas` font is used for all type names, property names and literals.
  - type names are in red with a light red background – `package`
  - property names are in bold style – **`created_at`**
  - literals (values) are in green with a green background – `IP Watchlist`
  - as “kinds of relationship” are string literals, they will also appear in green with a green background – `related-to`
- In property tables, if the property is being redefined from an inherited value in some way, then the background is dark grey.

All type names, property names and literals are in lower-case. Words in property names are separated with an underscore (_), while words in type names and string enumerations are separated with a dash (-).

6.5.2. Reserved Property Names

Reserved property names are marked with a type called `RESERVED` and a description text of “RESERVED FOR FUTURE USE”. Any property name that is marked as `RESERVED` **MUST NOT** be used in any implementation.

6.5.3. Examples

Examples are included, using the JSON Mandatory To Implement (MTI) serialization. They are in `Consolas` 9 pt font, with straight quotes, black text and a light blue background. JSON examples have a 2 space indentation.

6.6. Changes from STIX 1.x

< TO DO>

7. Common Types

Open questions:

1. Make sure each of the sections are consistent with each other once we're done
2. Need to figure out how to do statements about JSON MTI serialization

This section defines the common types used throughout STIX. These types will be referenced by the "Type" column in other sections. This section defines the names and allowable values of common types that are used in the STIX information model; it does not, however, define the meaning of any fields using these types. These types may be further restricted elsewhere in the document.

Type	Description
`boolean`	Contains a value of `true` or `false`.
`cybox-container`	A container for CybOX content. This type is defined by the [TODO CybOX Reference].
`external-reference`	Contains a non-STIX reference or identifier to other related content.
`identifier`	Contains an identifier (ID) for a STIX Object.
`kill-chain-phase`	Contains a reference to a kill chain phase by name.
`list`	Contains an ordered sequence of values. Often, the phrasing "`list` of type `<type>`" is used to indicate that all values within the list must conform to a specific type.
`maec-container`	A container for MAEC content. This type is defined by [TODO Ref

	MAEC].
`number`	Contains a number.
`object`	A container for generic data.
`string`	Contains text.
`timestamp`	Contains a timestamp (date and time).
`timestamp-precision`	Contains a precision value for timestamps.
`open-vocab`	Contains a value from a STIX open (`open-vocab`) or suggested vocabulary.
`controlled-vocab`	Contains a value from a STIX controlled (`controlled-vocab`) vocabulary.
`vocab-ext`	Contains a value for a field in an extended vocabulary.

7.1. Boolean

Type Name: `boolean`

A `boolean` contains a value of either true or false. Properties with this type **MUST** have a value of `true` or `false`.

The JSON MTI serialization uses the JSON boolean type, which is a literal (unquoted) `true` or `false`.

7.2. CybOX Container

Type Name: `cybox-container`

A container for CybOX content, as defined by [TODO Ref CybOX]. The structure is duplicated here simply for ease of use; normative usage is defined by the CybOX language.

<todo: When CybOX is finished, copy in>

7.3. External Reference

Type Name: `external-reference`

External references are used to describe pointers to information represented outside of STIX. For example, an incident could use an external reference to indicate an ID for that incident in an external database or a report could use references to represent source material.

7.3.1. Properties

Property Name	Type	Description
`source` (required)	`string`	The source within which the `external-reference` is defined (system, registry, organization, etc.).
`description` (optional)	`string`	A human readable description.
`url` (optional)	`url`	A URL reference to an external resource. [TODO: Reference to URL syntax]
`external_id` (optional)	`string`	An identifier for the external reference content.

7.3.2. Requirements

- At least one of **`external_id`**, **`url`**, and **`description`** fields **MUST** be present.

7.3.3. Examples

A `external-reference` from the CAPEC repository

```
{
  ...
  "external_references": [
    {
      "source": "capec",
      "external_id": "CAPEC-550"
    }
  ],
  ...
}
```

A external-reference from the CAPEC repository with URL

```
{
  ...
  "external_references": [
    {
      "source": "capec",
      "external_id": "CAPEC-550",
      "url": "http://capec.mitre.org/data/definitions/550.html"
    }
  ],
  ...
}
```

An external-reference to Mandiant's APT1 report document

```
{
  ...
  "external_references": [
    {
      "source": "Mandiant",
      "description": "APT1 report",
      "url": "http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf"
    }
  ],
  ...
}
```

An external-reference to a VERIS entry

```
{
  ...
  "external_references": [
    {
      "source": "veris",
      "external_id": "00C84D6A-CDB8-4A5B-A1A6-0D75A65274D7"
    }
  ],
  ...
}
```

An external-reference to a Jira item

```
{
  ...
  "external_references": [
    {
      "source": "jira",
      "external_id": "TAB-1370",
      "url": https://issues.oasis-open.org/browse/TAB-1370"
    }
  ],
  ...
```

```
}
```

7.4. Identifier

Type Name: `identifier`

An `identifier` uniquely identifies an instance of a STIX object. Identifiers **MUST** follow the form `[object-type]--[UUIDv4]`, where **`[object-type]`** is the exact value from the `type` field of the object being identified or referenced and where the **`[UUIDv4]`** is an RFC 4122-compliant Version 4 UUID. The UUID **MUST** be generated according to the algorithm(s) defined in RFC 4122, Section 4.4 (Version 4 UUID) [add reference].

7.4.1. Examples

```
{
  ...
  "type": "indicator",
  "id": "indicator--e2e1a340-4415-4ba8-9671-f7343fbf0836",
  ...
}
```

7.5. Kill Chain Phase

Type Name: `kill-chain-phase`

The `kill-chain-phase` represents a phase in a kill chain.

Property Name	Type	Description
`kill_chain_name` (required)	string	The name of the kill chain. The value of this field **SHOULD** be all lowercase (where lowercase is defined by the locality conventions) and **SHOULD** use dashes instead of spaces or underscores.
`phase_name` (required)	string	The name of the phase in the kill chain. The value of this field **SHOULD** be all lowercase (where lowercase is defined by the locality conventions) and **SHOULD** use

		dashes instead of spaces or underscores.

7.5.1. Examples

```
{
  ...
  "kill_chain_phases": [
    {
      "kill_chain_name": "kill-chain-foo",
      "phase_name": "phase-foo"
    }
  ],
  ...
}
```

7.6. List

Type Name: `list`

A `list` contains an ordered sequence of values. When the phrasing "`list` of type `<type>`" is used, all values in the list **MUST** be of the specified type. For instance, `list` of type `number` means that all values of the list must be of the number type. Upper and lower bounds of the list – the minimum and maximum number of elements – may be specified where the list is used. This section does not specify the upper and lower bounds of `list`.

The JSON MTI serialization uses the JSON array type, which is an ordered list of zero or more values.

3.6.2. Examples

```
{
  ...
  "observed_data_refs": [
    "observed-data--b67d30ff-02ac-498a-92f9-32f845f448cf",
    "observed-data--c96f4120-2b4b-47c3-b61f-eceaa54bd9c6",
    "observed-data--787710c9-1988-4a1b-9761-a2de5e19c62f"
  ],
  ...
}
```

7.7. MAEC

Type Name: `maec-container`

A container for MAEC content, as defined by [TODO Ref MAEC]. The structure is duplicated here simply for ease of use; normative usage is defined by the MAEC language.

[TODO: When MAEC 5.0 is finished, copy in].

7.8. Number

Type Name: `number`

The `number` type represents any number that can be expressed as a real number (e.g., -10, 0, 10, 10.1, 10.123213). Each use of `number` specifies the following:

- The valid range of values;
- Whether it is limited to integers or not; and
- The maximum number of decimal places, if non-integer values are permitted.

In the JSON MTI serialization, numbers are represented by the JSON number type.

7.8.1. Examples

```
{
  ...
  "count": 8,
  ...
}
```

7.9. Object

Type Name: `object`

The `object` data type represents a container for generic data. When used, the field description will have normative text describing the type of data in the field.

7.10. String

Type Name: `string`

The `string` data type represents a sequence of Unicode characters.

The JSON MTI serialization uses the JSON string type, which mandates the UTF-8 encoding to support Unicode.

7.10.1. Examples

```
{
  ...
  "title": "The Black Vine Cyberespionage Group",
  ...
}
```

7.11. Timestamp

Type Name: `timestamp`

The `timestamp` type defines how timestamps are represented in STIX. Most discrete timestamps (i.e., not time ranges or relative times) in STIX have a corresponding optional field that indicates the precision of the timestamp, of type `timestamp-precision`.

In cases where the timestamp is metadata about the STIX construct, such as creation and modification times for STIX Objects, the `timestamp` field will not have the corresponding precision field. In these cases, the precision field is `full`.

7.11.1. Requirements

- The `timestamp` field **MUST** be a valid RFC 3339-formatted timestamp [TODO add reference] using the format `YYYY-MM-DDTHH:mm:ss[.s+]Z` where the “s+” represents 0 or more sub-second values.
- The timestamp **MUST** be represented in the UTC timezone and **MUST** use the “Z” designation to indicate this.

7.11.2. Examples

A `timestamp` that does not have a precision field defined would look like:

```
{
```

```
  ...
  "created": "2016-01-20T12:31:12.12345Z",
  ...
}
```

7.12. Timestamp Precision

Type Name: `timestamp-precision`

A `timestamp-precision` represents the precision options for a given `timestamp`.

7.12.1. Requirements

- If present, the `timestamp-precision` field **MUST** have a value of `year`, `month`, `day`, `hour`, `minute`, or `full`.
    - The default value for the precision field is `full`, so omitting the field is equivalent to explicitly specifying `full`.
    - A value of `full` indicates that the value in the `timestamp` field is precise to the full number of digits in the timestamp value (including any fractional seconds, such as milliseconds or microseconds).
    - A value of `minute`, `hour`, `day`, `month`, or `year` indicates that the timestamp value is precise to that as a lower bound (the precision window is the timestamp value plus one unit of the precision value).
        - *For example, if the timestamp value is `2016-04-25T13:00:00Z` and the precision value is `hour`, the time is greater than or equal to `2016-04-25T13:00:00Z` and less than `2016-04-25T14:00:00Z`.*
    - When specifying a precision other than `full`, the time portion of the `timestamp` field **MUST** contain `00` for all fields beyond the specified precision while the date portion **MUST** contain `01` for all fields beyond the specified precision.
        - *For example, if the precision field is `month`, the `timestamp` field must contain `01` for the day field and `00` for the hour, minute, and second fields such as `2016-12-01T00:00:00Z`.*
- The `timestamp-precision` field will always be nested at the same level as the `timestamp` field.
- The property name for the precision field is **`[timestamp_field_name]_precision`**.
    - *For example, if the key of the `timestamp` field is **`created_at`**, the key of the precision field is **`created_at_precision`**.*

7.12.2. Examples

The following examples have explicitly defined the precision

A timestamp known only to a year would look like:

```
{
  ...
  "start": "2016-01-01T00:00:00Z",
  "start_precision": "year",
  ...
}
```

A timestamp known only to an hour would look like:

```
{
  ...
  "end": "2016-01-20T12:00:00Z",
  "end_precision": "hour",
  ...
}
```

The following examples have implicitly defined the precision
A timestamp known to a second would look like:

```
{
  ...
  "start": "2016-01-20T12:31:12Z",
  ...
}
```

A timestamp known to 5-digit sub-second precision would look like:

```
{
  ...
  "end": "2016-01-20T12:31:12.12345Z",
  ...
}
```

7.13. Open Vocabulary

Type Name: `open-vocab`

An open vocabulary is a string field that provides a list of suggested values, without constraining producers from extending those values. The list of suggested values is known as the suggested vocabulary. The value of an `open-vocab` field **SHOULD** be a value from the suggested vocabulary but **MAY** be any other `string` value. Values that are not from the value list **SHOULD** be all lowercase (where lowercase is defined by the locality conventions) and **SHOULD** use dashes instead of spaces or underscores.

7.13.1. Examples

Example
In this example the field indicator *labels* is an open vocabulary, which means any string value is valid, however, one should use a value from the suggested vocabulary..

```
{
  ...,
  "labels": ["malicious-activity"],
  ...
}
```

8. STIX Objects

<TODO> Add some description about STIX Objects here.

This section outlines the common aspects of all STIX Objects.Three types of STIX Objects have been defined in STIX 2.0:

1. STIX Domain Objects (SDOs) — 14 SDOs have been defined in this version;
2. Relationship Objects — three different types have been defined; and
3. Metadata Objects (`marking-definition`, `bundle`).

The STIX 2.0 information model is graph-based; graph nodes correspond to the 14 SDOs, and graph edges correspond to the three Relationship Objects.

8.1. Common Properties

This section defines properties and behaviors common to all STIX Domain Objects, Relationship Objects, and other special Metadata Objects.

Property Name	Type	Description
`type` (required)	`string`	The **`type`** property identifies the type of STIX Object (SDO, Relationship Object, etc). The value of the **`type`** field **MUST** be one of the types defined by a STIX Object (e.g., `indicator`).
`id` (required)	`identifier`	The **`id`** property uniquely identifies this object. All objects with the same **`id`** are

		considered different versions of the same object.
created_by_ref (optional)	`identifier`	The ID of the Source object that describes who created this object (`source--`).
labels (optional)	`list` of type `open-vocab`	This field specifies the type of object if known and allows for implementation dependant or trust group dependant labels or tags to also be applied to this object for further classification and sorting. This field usually includes a suggested vocabulary and items in this list **SHOULD** come from that vocabulary. Additional labels **MAY** be added beyond what is in the open / suggested vocabulary based on needs and requirements of implementations and trust groups.
version (required)	`number`	The **version** property indicates the version of this object. This field's value **MUST** be an integer (whole number) greater than or equal to 1 and less than or equal to 999,999,999. Higher numbers indicate later versions of the object. Object creators **MUST** increase the version number (**SHOULD** increment it by exactly 1) when creating a new version of an object.
created (required)	`timestamp`	The **created** property represents the time at which the first version of this object was created. The object creator **SHOULD** use the time it deems most appropriate as the time the object was created. The **created** property **SHOULD** be the same across all versions of the object unless the **created** was corrected by some version.

modified (required)	`timestamp`	The **modified** property represents the time that this particular version of the object was created. The object creator **SHOULD** use the time it deems most appropriate as the time this version of the object was created. The **modified** for a given object version **MUST** be greater than or equal to the **created**.
revoked (optional)	`boolean`	The **revoked** property indicates whether the object has been revoked. Revoked objects are no longer considered worthwhile intelligence. Revoking an object is permanent; future versions of the object with this **id** **MUST NOT** be created.
version_comment (optional)	`string`	A comment outlining why the new version of this object was created.
external_references (optional)	`list` of type `external-reference`	A list of external references which refers to non-STIX information. This field **MAY** be used to provide one or more URLs, descriptions, or IDs to records in other systems.
confidence (reserved)	`RESERVED`	RESERVED FOR FUTURE USE
object_marking_refs (optional)	`list` of type `identifier`	The list of `marking-definition` objects to be applied to this object. When multiple markings of the same type appear in this list, the markings appearing later have precedence over those appearing earlier.
granular_markings (optional)	`list` of type `granular-marking`	The set of granular markings applied to this object.

8.2. IDs and References

The **id** field uniquely identifies an object. It **MUST** conform to the `identifier` type.

The STIX language makes use of globally unique identifiers as defined by the `identifier` type for all STIX Objects. The `identifier` type is also used to define fields that are *ID references* to

other constructs (such as the **created_by_ref** field in all STIX Objects). *Resolving* an ID reference is the process of identifying and obtaining the actual object referred by the ID reference field. ID references resolve to an object when the value of the ID reference field (e.g., **created_by_ref**) is an exact match with the **id** field of another object. ID references **MAY** refer to objects to which the consumer may not currently have access.

8.3. Object Creator

The `identifier` of the object creator is stored in the **created_by_ref** field, capturing the Source of the creator. The object creator is the entity (e.g., system, organization, instance of a tool) that generates the **id** field for a given object.

Entities that re-publish an object from another entity without making any changes to the object, and thus maintaining the original **id**, are not considered the object creator and **MUST NOT** change the **created_by_ref** field. Entities that accept objects and republish them with modifications or omissions **MUST** create a new **id** for the object and update the **created_by_ref** field to reflect them as an Source as they will be considered the object creator of the new object.

8.4. Versioning

STIX content is versioned using the **version**, **version_comment**, **revoked**, **created**, and **modified** properties. See the properties table in Section TODO [add reference] for full definitions and normative usage of the individual properties; this section describes the broader versioning process and normative rules for performing versioning and revocation.

STIX Objects can be versioned in order to update, add, or remove information. All STIX representations of objects are actually representations of a version of that object, identified in the **version** field. Higher values in the **version** field indicate later versions of the object. STIX Objects have a single *object creator*: the entity that generates the **id** for the object and creates the first version. Only the object creator is permitted to create new versions of a STIX Object. Producers other than the object creator **MUST NOT** create new versions of that object.

Individual versions of STIX Objects are immutable: representations of a given version of an object (identified by the object's **id** and its **version**) **MUST**, in all representations, always have the same set of properties and the same values for each property. In order to change the value of any property, or to add or remove properties, the **version** number **MUST** be increased to indicate a new version.

Objects can also be revoked, which is an indication that they are no longer considered actionable or worthwhile intelligence. As with issuing a new version, only the object creator is permitted to revoke a STIX object. A value of `true` in the **`revoked`** field indicates that an object has been revoked. Revocation is permanent: once an object is marked as revoked, later versions of that object **MUST NOT** be created. The change to the revoked field to indicate that an object is revoked is an update to object, and therefore the revoked object **MUST** have an updated **`version`** and **`modified`**.

8.4.1. Versioning Timestamps

Non-Normative Section

There are two timestamps used to indicate when STIX Objects were created and modified: **`created`** and **`modified`**. The **`created`** field indicates the time the first version of the object was created. The **`modified`** field indicates the time the current version of the object (indicated by the value of the **`version`** field) was created. For both **`created`** and **`modified`**, object creators may use any time that best represents when they want to say the object was created and modified. Some creators might use the time the representation was created in their own database; other creators might use the time the object was first shared externally; and others might use a user-tunable value.

8.4.2. New Version or New Object?

Non-Normative Section

Eventually an implementation will encounter a case where a decision must be made regarding whether a change is a version of an existing object or is different enough that it is a new object. This is generally considered a data quality problem and therefore this specification does not provide any normative text.

However, to assist implementers and promote consistency across implementations, some rules of thumb are provided. Any time a change indicates a *material change* to the meaning of the object (say, a different malware or a different actor), a new object with a different **`id`** should be used. The creator should also think about references to the object when deciding if a change is material. If the change would invalidate the references to the object, then the change is material and a new object **`id`** should be used.

8.4.3. Examples

Example Version

One object creator has decided that the previous title they used for a SDO is incorrect. They consider that change an update to the object.

Step #	STIX Object	Object Creator Action

1	<pre>{
	"type": "example",
	"id": "example-1",
	"created": "2016-05-01T06:13:14.000000Z",
	"modified": "2016-05-01T06:13:14.000000Z",
	"version": 1,
	"title": "attention",
	"description": "this is the description"
	}</pre>
		Original version of an object is created.

2	<pre>...
	"title": "Attention!",
	...</pre>
		Object creator changes the title.

3	<pre>{
	"type": "example",
	"id": "example-1",
	"version": 2,
	"created": "2016-05-01T06:13:14.000000Z",
	"modified": "2016-05-08T03:43:44.000000Z",
	"title": "Attention!",
	"description": "this is the description"
	}</pre>
		Object creator increases current object **version** by 1 and updates the **modified**.

Example of Derived Object

One object creator has decided that the previous title they used for a SDO is incorrect. They consider that change fundamental to the meaning of the object and therefore revoke the object and issue a new one.

1. A revised initial indicator, with the same **`id`**, updated **`version`**, and the **`revoked`** flag set.
2. A new indicator with a new **`id.`**

Step #	STIX Object	Object Creator Action

1	<pre>{
	"type": "example",
	"id": "example-1",
	"version": 1,
	"created": "2016-05-01T06:13:14.000000Z",
	"modified": "2016-05-01T06:13:14.000000Z",
	"title": "attention",
	"description": "this is the description"
	}</pre>
		Original object created (via new id and set **version** to *1*).

2	... "title": "Attention!", ...	Object creator changes the title.

3	{
	"type": "example",
	"id": "example-1",
	"version": 1,
	"created": "2016-05-01T06:13:14.000000Z",
	"modified": "2016-05-08T03:43:44.000000Z",
	"title": "attention",
	"description": "this is the description",
	"revoked": true
	}
		Object creator revokes the existing object by setting **revoked** to true.

4	{
	"type": "example",
	"id": "example-2",
	"version": 1,
	"created": "2016-05-08T03:43:44.000000Z",
	"modified": "2016-05-08T03:43:44.000000Z",
	"title": "Attention!",
	"description": "this is the description"
	}
		Object creator creates a new object (via new **id** and set **version** to *1*).

Example Recipient Workflow

This section describes an example workflow where a recipient receives multiple updates to a particular object series. (In this example, the STIX Objects have been truncated for brevity.)

Step #	STIX Object	Recipient Action

1	{
	"type": "example",
	"id": "example-1",
	"version": 1,
	"created": "2016-05-01T06:13:14.000000Z",
	"modified": "2016-05-01T06:13:14.000000Z"
	}
		Recipient stores example object because this is the first time the recipient has seen the object.

2	{
	"type": "example",
	"id": "example-1",
	"version": 4,
	"created": "2016-05-01T06:13:14.000000Z",
	"modified": "2016-05-08T03:43:44.000000Z"
	}
		Recipient updates example object because the received version number is higher than the object that is currently stored.

3	{
	"type": "example",
	"id": "example-1",
	"version": 3,
	"created": "2016-05-01T06:13:14.000000Z",
	"modified": "2016-05-06T06:23:45.000000Z"
	}
		Recipient ignores this object because the recipient already has a newer version of the object.
		Note: recipient might choose to store meta-information about received objects, including versions that were received out-of-order.

4	{
	"type": "example",
	"id": "example-1",
	"version": 12,
	"revoked": true,
	"created": "2016-05-01T06:13:14.000000Z",
	"modified": "2016-05-11T06:41:21.000000Z"
	}
		Recipient deletes example object, but keeps some metadata regarding the object.

5	{
	"type": "example",
	"id": "example-1",
	"version": 11,
	"created": "2016-05-01T06:13:14.000000Z",
	"modified": "2016-05-10T17:28:54.000000Z"
	}
		Recipient ignores this object because the recipient already has a newer version of the object (the revoked version).

Example Object Creator Workflow

This section describes an example workflow where a object creator publishes multiple updates to a particular object series. This scenario assumes a human using a STIX implementation. (In this example, the STIX Objects have been truncated for brevity.)

Step #	User Action	STIX Object
1	User clicks a create button in the user interface, creates a SDO, then clicks save. This action causes information to be stored in the product's database.	N/A – STIX is not involved in this scenario. (Tools *could* choose to create and track STIX versions for internal changes, but it is not required by the specification.)
2	The user clicks the "share" button, delivering the intelligence to sharing partners.	{ "type": "example", "id": "example-2", "version": 1,

		<pre> "created": "2016-05-01T06:13:14.000000Z", "modified": "2016-05-01T06:13:14.000000Z" }</pre>
3	The user performs additional analysis within the STIX implementation, performing multiple modifications and saving their work multiple times.	N/A – STIX is not involved in this scenario. (Tools *could* choose to create and track STIX versions for internal changes, but it is not required by the specification.)

4	The user, happy with the status of their work, decides to provide an update to the previously published object.	<pre>{
		  "type": "example",
		  "id": "example-2",
		  "version": 2,
		  "created": "2016-05-01T06:13:14.000000Z",
		  "modified": "2016-05-03T16:33:51.000000Z"
		}</pre>

5	The user receives lots of negative feedback regarding the quality of their work and decides to retract the object by pressing the “revoke” button.	<pre>{
		  "type": "example",
		  "id": "example-2",
		  "version": 3,
		  "revoked": true,
		  "created": "2016-05-01T06:13:14.000000Z",
		  "modified": "2016-05-08T13:35:12.000000Z"
		}</pre>

8.5. Common Relationships

Common Relationships are relationships that are defined for all STIX Objects. See Section <to do>[add reference] for more information about relationships.

Name	Source	Target	Description
`duplicate-of`	*`<Object>`*	*`<Object of same type>`*	Allows recording that two different Objects of the same type are duplicates of each other and one should be disregarded or deleted.
`related-to`	*`<Object>`*	`*`	The catch-all generic relationship type that allows description of relationships between an Object and

			any other Domain Object. It is essentially a fallback to allow a relationship to be defined when none of the others fit.

8.6. Reserved Properties

This section defines property names that are reserved for future use in revisions of this document. The property names defined in this section **MUST NOT** be used for the name of any Custom Property.

Global properties are:

- `confidence`
- `severity`

Specific properties per SDO are:

- Course of Action SDO: `action`
- Source, Threat Actor, Victim SDOs: `usernames`, `phone_numbers`, `addresses`

Open Questions:

- How should confidence be defined? What's the scale?
- Should confidence be required or optional (may be different in different locations)?
- Is it a controlled vocab with _ext fields or just an enum

9. Data Handling in STIX

Data markings provide the ability to mark data in STIX, typically to represent restrictions and permissions for how that data can be used and shared. For example, data may be shared with the restriction that it must not be re-shared, or that it must be encrypted at rest. In STIX, data markings are specified using the `marking-definition` object. Object markings and granular markings apply those to objects and fields, respectively.

Some types of marking definitions or trust groups have rules about which markings have precedence over other markings or which markings can be additive to other markings. The STIX specification does not define precedence or which marking definitions should have precedence over other definitions.

9.1. Marking Definition

Type Name: `marking-definition`

The `marking-definition` object represents a specific instance of a marking, defined by an external marking system (e.g., the traffic light protocol, TLP) and contained in the definition field. Data markings typically capture handling or sharing requirements for data, and are applied in the **`object_markings_refs`** field that is optional on any STIX Object, which references a list of IDs for `marking-definition` objects. If marking definitions are themselves marked, they **MUST** be marked with another marking definition, not marked with themselves.

The **`confidence`** field is included in the `marking-definition` for the sake of consistency however this property **SHOULD NOT** be used for `marking-definition` objects.

9.1.1. Properties

Common Properties		
`type, id, created_by_ref, labels, version, created, modified, revoked, version_comment, external_references, confidence, object_markings_refs, granular_markings`		
Property Name	**Type**	**Description**
`type` (required)	`string`	The value of this field **MUST** be `marking-definition`.
`definition` (required)	`object`	The detailed information of the marking. This field **SHOULD** be of type: `tlp-marking` or `statement-marking`. Other marking definitions **MAY** be used.

9.1.2. Relationships

These are no relationships explicitly defined between the Marking Definition object and other objects, other than those defined as common relationships. The first section lists the embedded relationships by property name along with their corresponding target.

Relationships are not restricted to those listed below. Relationships can be created between any objects using the `related-to` relationship name or, as with open vocabularies, user-defined names.

Embedded Relationships	
created_by_ref	source
object_markings_refs	marking-definition
Common Relationships	
duplicate-of, related-to	

9.1.3. TLP Marking Type

The TLP marking type sub-object defines how you would represent a TLP marking in a definition field. TLP markings with higher precedence override those with lower precedence.

Property Name	Type	Description
type (required)	string	The value of this field **MUST** be tlp-marking.
tlp (required)	tlp-marking-enum	The TLP level (defined by FIRST, ask Tom Millar for stable ref) of the content marked by this marking definition.

The following standard definitions **MUST** be used to reference or represent TLP markings:

white	{
	"type": "marking-definition",
	"id": "marking-definition--613f2e26-407d-48c7-9eca-b8e91df99dc9",
	"created": "2016-08-01T00:00:00Z",
	"modified": "2016-08-01T00:00:00Z",
	"version": 1,
	"definition": {
	"type": "tlp-marking",
	"tlp": "white"
	}
	}

green	{ "type": "marking-definition", "id": "marking-definition--34098fce-860f-48ae-8e50-ebd3cc5e41da", "created": "2016-08-01T00:00:00Z", "modified": "2016-08-01T00:00:00Z", "version": 1,

	"definition": {
	"type": "tlp-marking",
	"tlp": "green"
	}
	}

amber	{
	"type": "marking-definition",
	"id": "marking-definition--f88d31f6-486f-44da-b317-01333bde0b82",
	"created": "2016-08-01T00:00:00Z",
	"modified": "2016-08-01T00:00:00Z",
	"version": 1,
	"definition": {
	"type": "tlp-marking",
	"tlp": "amber"
	}
	}

red	{
	"type": "marking-definition",
	"id": "marking-definition--5e57c739-391a-4eb3-b6be-7d15ca92d5ed",
	"created": "2016-08-01T00:00:00Z",
	"modified": "2016-08-01T00:00:00Z",
	"version": 1,
	"definition": {
	"type": "tlp-marking",
	"tlp": "red"
	}
	}

9.1.4. Statement Marking Type

The statement marking type sub-object defines how you would represent a text marking statement (copyright, terms of use, etc.) in a definition. Statement marking types do not override each other.

Property Name	Type	Description
type (required)	`string`	The value of this field **MUST** be `statement-marking`.
statement (required)	`string`	A statement (e.g., copyright, terms of use) applied to the content marked by this marking definition.

9.1.5. Examples

```
{
  "type": "marking-definition",
  "id": "marking-definition--34098fce-860f-48ae-8e50-ebd3cc5e41da",
  "created": "2016-08-01T00:00:00Z",
  "modified": "2016-08-01T00:00:00Z",
  "version": 1,
  "definition": {
    "type": "tlp-marking",
    "tlp": "green"
  }
}

{
  "type": "marking-definition",
  "id": "marking-definition--089a6ecb-cc15-43cc-9494-767639779124",
  "created": "2016-08-01T00:00:00Z",
  "modified": "2016-08-01T00:00:00Z",
  "version": 1,
  "definition": {
    "type": "statement-marking",
    "statement": "Copyright 2016, Example Corp"
  }
}

{
  "type": "indicator",
  "id": "indicator--089a6ecb-cc15-43cc-9494-767639779235",
  ...
  "object_marking_refs": [
    "marking-definition--089a6ecb-cc15-43cc-9494-767639779124",
    "marking-definition--34098fce-860f-48ae-8e50-ebd3cc5e41da"
  ],
  ...
}
```

9.2. Object Markings

Object Markings define how markings are applied to STIX objects (SDOs and Relationship Objects). Object Markings are contained in the **`object_marking_refs`** field, which is an optional list of ID references (of type `identifier`) that resolve to objects of type `marking-definition`. The markings referenced by the **`object_marking_refs`** field and defined in the `marking-definition` object apply to that TLO and all of its fields. Changes to the **`object_marking_refs`** field (and therefore the markings applied to the object) are treated

the same as changes to any other properties on the object and follow the same rules for versioning.

9.2.1. Examples

This example marks the indicator with the marking definition referenced by the ID.

```
{
  "type": "indicator",
  "id": "indicator--089a6ecb-cc15-43cc-9494-767639779235",
  ...
  "object_marking_refs": ["marking-definition--089a6ecb-cc15-43cc-9494-767639779123"],
  ...
}
```

9.3. Granular Markings

Granular Markings define how markings are applied to individual portions of STIX objects (SDOs and Relationship Objects). Granular Markings are contained in the **granular_markings** field, which is a list of `granular-marking` instances. Each of those instances contains a list of selectors to indicate what is marked and a list of references to the `marking-definition` objects to be applied. Granular Markings can be used, for example, to indicate that the **name** field of an `indicator` should be handled as TLP:GREEN, the **description** as TLP:AMBER, and the **pattern** as TLP:RED.

9.3.1. Granular Marking Type

The `granular-marking` type defines how the list of `marking-definition` objects referenced by the **marking_refs** property to apply to a set of content identified by the list of selectors in the **selectors** property.

Property Name	Type	Description
marking_refs (required)	`list` of type `identifier`	The list of markings that apply to the fields selected by **selectors**.
selectors (required)	`list` of type `string`	A list of selectors for content contained within the STIX object in which this field appears. Selectors **MUST** conform to the syntax defined in [Section TODO]. The markings referenced in the **marking_refs** field are applied to the content selected by the selectors in this list.

9.3.1.1. Selector Syntax

Selectors contained in the **`selectors`** list are strings that consist of multiple components that **MUST** be separated by the `.` character. Each component **MUST** be one of:

- A property name, e.g. `description`, or;
- A zero-based array index, specified as a non-negative integer in square brackets, e.g. `[4]`

Selectors are path traversals: the root of each selector is the STIX object that the **`granular_markings`** field appears in. Starting from that root, for each component in the selector, properties and array items are traversed. When the complete list has been traversed, the value of the content is considered selected.

Selectors **MUST** refer to properties or array items that are actually present on the marked object.

As an example, consider the following STIX object:

```
{
  "type": "example",
  "description": "hello",
  "extra": {
     "foo": "some field",
     "bar": "some other field"
  },
  "widgets": ["first", "second"]
}
```

Valid selectors:

- `description` selects the **`description`** property ("`hello`")
- `extra.foo` selects the **`foo`** property contained within the **`extra`** property ("`some field`")
- `widgets.[0]` selects the first item contained within the **`widgets`** list ("`first`")
- `widgets` selects the list contained in the **`widgets`** property. Due to the recursive nature of the selector, that includes all items in the list (`["first", "second"]`).
- `extra` selects the object contained in the **`extra`** property. Due to the recursive nature of the selector, that includes both the **`foo`** and **`bar`** properties.

Invalid selectors:

- `foobar` and `widgets.[3]` are invalid selectors because they refer to content not present in that object.
- `extra.[0]` is an invalid selector because the `extra` property is an object and not a list.
- `widgets.name` is an invalid selector because `widgets` property is a list and not an object.

Non-Normative Text
This syntax is inspired by JSONPath and is in fact a strict subset of allowable JSONPath expressions (with the exception that the '$' to indicate the root is implicit). Care should be taken when passing selectors to JSONPath evaluators to ensure that the root is correct. It is expected, however, that selectors can be easily evaluated in programming languages that implement list and key/value mapping types (dictionaries, hashmaps, etc.) without resorting to an external library.

9.3.2. Example

This example marks the description property with the single marking definition referenced in the list.

```
{
  ...
  "granular_markings": [
    {
      "selectors": ["description", "entries"],
      "marking_refs": ["marking-definition--089a6ecb-cc15-43cc-9494-767639779123"]
    }
  ],
  "description": "Some description"
  "title": "Some title",
  "entries": ["first", "second"]
}
```

10. Customizing STIX

10.1. Custom Properties

The authors of this specification recognize that there will be cases where certain information exchanges can be improved by adding fields that are not specified nor reserved in this document; these fields are called **Custom Properties**. This section provides guidance and requirements for how producers can use Custom Properties and how consumers should interpret them in order to extend STIX in an interoperable manner.

10.1.1. Requirements

- A STIX object **MAY** have any number of Custom Properties.
- Custom Properties **MUST** be in ASCII and are limited to characters a-z (lowercase ASCII) and underscore (_).
- Custom Properties **SHOULD** start with “x_” followed by a source unique identifier (like a domain name), an underscore and then the name. For example: **`x_examplecom_customfield`**.
- Custom Property keys **SHOULD** be no longer than 30 ASCII characters in length.
- Custom Property keys **MUST** have a minimum length of 3 ASCII characters.
- Custom Property keys **MUST** be no longer than 256 ASCII characters in length.
- Custom Properties that are not prefixed with “x_” may be used in a future version of the specification for a different meaning. If compatibility with future versions of this specification is required, the “x_” prefix **MUST** be used.
- Custom Properties **SHOULD** be uniquely named when produced by the same source and **SHOULD** use a consistent namespace prefix (e.g., a domain name).
- Custom Properties **SHOULD** only be used when there is no existing field defined by the STIX specification that fulfills that need.

A consumer that receives a STIX document with one or more Custom Properties it does not understand **MAY** refuse to process the document further, or silently ignore non-understood properties and continue processing the document.

The reporting and logging of errors originating from the processing of Custom Properties depends heavily on the technology used to transport the STIX document and is therefore not covered in this specification.

Non-Normative: Producers of STIX documents that contain Custom Properties should be aware of the variability of consumer behavior depending on whether or not the consumer understands the Custom Properties present in a STIX object. Rules for processing Custom Properties should be well defined and accessible to any consumer that would be reasonably expected to parse them.

10.1.2. Examples

```
{
  ...,
  "x_acmeinc_scoring": {
    "impact": "high",
    "probability": "low"
  },
  ...
}
```

10.2. Custom Objects

The authors of this specification recognize that there will be cases where certain information exchanges can be improved by adding objects that are not specified nor reserved in this document; these objects are called **Custom Objects**. This section provides guidance and requirements for how producers can use Custom Objects and how consumers should interpret them in order to extend STIX in an interoperable manner.

10.2.1. Requirements

- Producers **MAY** include any number of Custom Objects in STIX content.
- Custom Objects **MUST** contain the required Common Properties (**`id`**, **`type`**, **`version`**, **`modified`**, **`created`**, **`created_by_ref`**) and **MAY** contain any optional Common Property (defined in Section TODO).
    - The definitions of these properties are the same as those defined in Common Properties and therefore those fields **MUST NOT** be used to represent the custom properties in the object.
- The **type** field in a Custom Object **MUST** be in ASCII and are limited to the characters a-z (lowercase ASCII) and hyphen (-).
- Custom Object names **SHOULD** be no longer than 30 ASCII characters in length.
- Custom Object names **MUST** have a minimum length of 3 ASCII characters.
- Custom Object names **MUST** be no longer than 256 ASCII characters in length.
- The value of the **`type`** field in a Custom Object **SHOULD** start with “x-” followed by a source unique identifier (like a domain name), a dash and then the name. For example: `x-examplecom-customobject`.
- A Custom Object whose name is not prefixed with “x-” may be used in a future version of the specification with a different meaning. Therefore, if compatibility with future versions of this specification is required, the “x-” prefix **MUST** be used.
- The value of the **`id`** field in a Custom Object **MUST** use the same format as the `identifier` type, namely, name--uuid
- Custom Objects **SHOULD** be uniquely named when produced by the same source and **SHOULD** use a consistent namespace prefix (e.g., a domain name).
- Custom Objects **SHOULD** only be used when there is no existing SDO or Relationship Object defined by the STIX specification that fulfills that need.

A consumer that receives a STIX document with one or more Custom Objects that it does not understand **MAY** refuse to process the document further, or silently ignore non-understood objects and continue processing the document.

The reporting and logging of errors originating from the processing of Custom Objects depends heavily on the technology used to transport the STIX document and is therefore not covered in this specification.

Non-Normative Section
Producers of STIX documents that contain Custom Objects should be aware of the variability of consumer behavior depending on whether or not the consumer understands the Custom Objects. Rules for processing Custom Objects should be well defined and available to any consumer that would be reasonably expected to parse them.

10.2.2. Examples

```
{
  "type": "bundle",
  "id": "bundle--f37aa79d-f5f5-4af7-874b-734d32c08c10",
  "custom_objects": [
    {
      "type": "x-examplecom-customobject",
      "id": "x-examplecom-customobject--4527e5de-8572-446a-a57a-706f15467461",
      "created": "2016-08-01T00:00:00Z",
      "modified": "2016-08-01T00:00:00Z",
      "version": 1,
      "some_custom_stuff": 14,
      "other_custom_stuff": "hello"
    }
  ]
}
```