STIX 2.0 Specification - Pre-Draft

STIX Core Concepts - Version 0.1

Document Table of Contents

1. Document Development Status

Object / Concept	Status	MVP	Description
STIX Concepts			
ID References	Draft	Yes	< add text >
Versioning	Development	Undecided	< add text >
Controlled Vocabularies	Review	Yes	< add text >
ENUMs	Concept	Undecided	< add text >
Vendor Defined Fields	Draft	Yes	< add text >
Data Markings	Review	Yes	< add text >
Core Types			
`identifier`	Draft	Yes	< add text >
`timestamp`	Draft	Yes	< add text >
`vocab-ext`	Review	Yes	< add text >
`granular-marking`	Review	Undecided	< add text >

Common Properties			
`stix-core`	Development	Yes	< add text >
`descriptive-properties`	Development	Yes	< add text >
Common Types			
`impact`	Concept	Undecided	< add text >
`statement`	Concept	Undecided	< add text >

2. STIX Definitions

Term	Definition
top level object	Top level objects (TLOs) capture information about things that exist in the cybersecurity threat domain: threat actors, observations, attack patterns, malware, campaigns, etc.
ID reference	A string that contains an ID that references a different existing STIX TLO.

3. STIX Concepts

3.1. IDs and ID References

	Status: Draft **MVP**: Yes

The CTI language specifications make use of globally unique identifiers as defined by the `identifier` type. The type is also used to define fields that are *references* to other constructs (such as the **`source_ref`** field in relationship). *Resolving* a reference is the process of identifying and obtaining the actual object referred to by the reference field. References resolve to an object when the value of the reference field (e.g. **`source_ref`**) is an exact match with the **`id`** field of another object. References may refer to any object that exist - even those that the consumer may not currently have access to.

3.2. Versioning

	Status: Development **MVP**: Undecided

The following table documents the set of use cases that drive the requirements on versioning within STIX.

Open Questions:

1. Need to define what a material change is, since there is a MUST with it, we need to make sure the definition is solid.
2. Do we need the ability to revoke a specific version of the object?
3. Do we need to better define "entity" in object creator? Is it an organization, a system, something else? The intent was to leave it open to any, is that OK?
4. Is there a better name for the revision field?
5. Do we need any timestamp fields? Which ones?
6. What are the versioning-specific relationships? Do we need derived-by, suggested-update, etc.?
7. May need an FAQ section at some point, this is a complicated topic

3.2.1. Use Cases

Uses we consider in-scope for versioning in STIX 2.0 MVP.

1. Object creator creates a new top-level object.
    a. Object creator creates a new top-level object derived from a top-level object from another object creator.
    b. Object creator creates a new top-level object derived as a major change from an top-level object from the same object creator.
    c. Object creator creates a new top-level object not based on any other object.
2. Object creator updates a top-level object.
    a. Object creator updates a top-level object with additional content.
    b. Object creator updates a top-level object with changes to existing content. <-- previously issued intel contained valid content that is now refined/improved
    c. Object creator updates a top-level object with corrections to existing content. <-- previously issued intel contained errors that are now corrected
3. Object creator indicates that a top-level object and all of its versions are no longer "valid" <-- previously issued intel was incorrect and an update will not be issued.

3.2.2. Scenarios

Scenarios that demonstrate the above use cases (numbers align)

1a. ACME Incorporated receives an indicator with a single IP address. The indicator includes a relationship to an indicated TTP for a particular piece of malware. (3 objects total). ACME Incorporated creates a new indicator with the single IP address received in the original indicator but adds an OR hash=x to the pattern. It creates a new relationship from the new indicator to the previously indicated TTP. It creates a "derived_from" relationship from the new indicator to the original received indicator. (4 objects total; 5 if you count the old original indicator)

1b. Major Financial creates a TA asserting a title (name), motivation, sophistication and some identification details. It realizes later learns more about the threat actor that will change most of the information in the TA characterizing that threat actor. It creates a new TA with the updated information. It creates a "derived_from" relationship from the new TA to the old TA.

1c. Government Org Alpha creates a new threat actor characterization for Deep Baboon, shares it to several other government organizations.

2a. Major Financial updates the previous TA, adding an assertion of intended_effects.

2b. ACME Incorporated updates the previous indicator, adding a couple new IP addresses to the pattern.

2c. ACME Incorporated updates the original indicator in 1a above and changes the single IP address making explicit that the previous indicator was in error and that the new version is corrected.

2d. GOA expands attribution for a threat actor, adding a more specific location.

3a. ACME Incorporated revokes the indicator in 1a, it incorrectly listed the Google DNS server.

3b. GOA revokes the threat actor, they had an incorrect country. The re-issue an update with Country Bravo rather than Country Delta.

3.2.3. Implementation

This section describes versioning in STIX. Within STIX, versioning is controlled by the fields, as described below.

3.2.4. Definitions

Object Creator: The object creator is the entity (e.g. system, organization, tool) that generates the `identifier` field for a given object. Entities (proxies) that re-publish a top-level object from another entity, maintaining the original `identifier`, are not considered the object creator. Entities (brokers) that accept objects and republish them with a new identifier are considered the object creator of the new objects.

Top-Level Object: TODO LINK TO DEF

Object Series: a top-level object series is the set of STIX top-level objects with the same ID. All objects are members of a top-level object series. a top-level object series may only have a single member.

Property Name	Type	Description
`revision` (required)	`number`	**`revision`** indicates the revision number of this object. This field **MUST** be present in all STIX objects. This field's value **MUST** be greater than or equal to 1 and less than or equal to 999,999,999. Higher revision numbers indicate later versions of the object. Object creators **MUST** increment the revision number (**SHOULD** increment it by exactly 1) when creating a new version of a top-level object.
`revoked` (optional)	`boolean`	**`revoked`** indicates whether this object has been revoked. If this field is present, the timestamp indicates the timestamp that the object was revoked. Revoking a top-level object terminates the complete object series (all versions of the object); future revisions of that object are not permitted. When revoking a top-level object, the revision number **MUST** be incremented as above.

STIX objects, uniquely identified by **`id`** and **`revision`** values, are considered to be immutable (this means that the contents (fields and values) of the object **MUST NOT** be changed). Versioning provides the mechanism to change STIX objects.

STIX objects with the same **`id`** value and different **`revision`** values are said to be of the same "object series". Objects with different **`id`** values are said to be of different "object series". Within a top-level object series, lower **`revision`** values indicate "earlier" revisions, and higher **`revision`** values indicate later revisions.

If the **`revoked`** field is present in a top-level object, that object is said to have been "revoked". If the **`revoked`** field is not present, the object is said to be "active". Revocation terminates a top-level object series. Once a top-level object has been revoked, additional updates to the object series are not permitted. object creators **MUST NOT** publish new revisions to a top-level object series once that object series has been terminated.

Relationships may have sources or targets to object series' that have been revoked. The consumer may choose to handle those relationships however it wishes.

New Object or Revision?
Eventually, an implementation will encounter a case where a decision must be made regarding whether a change is a new object series or a revision to an existing object series. 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.

Anytime a change indicates a material change to the meaning of the object (say different malware, different actor) a new object id **MUST** be used. The determination of whether a change is a material change is at the object creator's discretion.

3.2.5. Examples

Example Revision
Given: One object creator has decided that for their content, a change to the list of IP addresses in network indicators does not constitute a material change.

That object creator would consider each change of IP address to be an update to that indicator, and when changing the indicator would update the **`revision`** but not issue a new **`id`**.

Step #	STIX Object	Object Creator Action

1	<pre>{
	"type": "example",
	"id": "example-1",
	"revision": 1,
	"title": "attention",
	"description": "this is the description"
	}</pre>
		Original object created.

2		Object creator changes the title.

3	<pre>{
	"type": "example",
	"id": "example-1",
	"revision": 2,
	"title": "Attention!",
	"description": "this is the description"
	}</pre>
		Object creator increases current object **revision** by 1.

Example of Derived Object

Given: A different object creator has decided that for their content, a change to the list of IP addresses in network indicators does constitute a material change.

That object creator would then issue two objects for each update:

- A revised initial indicator, with the same **`id`**, updated **`revision`**, and the **`revoked`** flag set.
- A new indicator with a new **`id`**

Step #	STIX Object	Object Creator Action

1	<pre>{
	"type": "example",
	"id": "example-1",
	"revision": 1,
	"title": "attention",
	"description": "this is the description"
	}</pre>
		Original object created (via new id and set **revision** to *1*).

2		Object creator changes the title.

3	<pre>{
	"type": "example",
	"id": "example-2",
	"revision": 1,
	"title": "Attention!",
	"description": "this is the description"
	}</pre>
		Object creator creates a new object (via new **id** and set **revision** to *1*).

Example Recipient Workflow

This section describes an example workflow where a recipient receives multiple updates to a particular object series. The STIX Objects have been truncated for brevity.

Step #	STIX Object	Recipient Action

1	<pre>{
	"type": "example",
	"id": "example-1",
	"revision": 1
	}</pre>
		Recipient stores example object because this is the first time the recipient has seen the object.

2	<pre>{
	"type": "example",
	"id": "example-1",
	"revision": 4
	}</pre>
		Recipient updates example object because the received revision number is higher than the object that is currently stored.

3	{
	  "type": "example",
	  "id": "example-1",
	  "revision": 3
	}
		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 revisions that were received out-of-order.

4	{
	  "type": "example",
	  "id": "example-1",
	  "revision": 12,
	  "revoked": "2016-03-11T07:23:00Z"
	}
		Recipient deletes example object, but keeps some metadata regarding the object.

5	{
	  "type": "example",
	  "id": "example-1",
	  "revision": 11
	}
		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. The STIX Objects have been truncated for brevity. This scenario assumes a human using a STIX implementation.

Step #	User Action	STIX Object
1	User clicks a create button in the UI, creates a top-level object, 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 revisions for internal changes, but it is not required by the specification)

2	The user clicks the "share" button, delivering the latest cutting-edge threat intel to multiple social media platforms using STIX.	{
		  "type": "example",
		  "id": "example-2",
		  "revision": 1
		}

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 revisions 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.	{
		"type": "example",
		"id": "example-2",
		"revision": 2
		}

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.	{
		"type": "example",
		"id": "example-2",
		"revision": 3,
		"revoked": "2016-03-11T07:23:00Z"
		}

Example: Broker

A broker is an entity that accepts STIX objects from entities wants to become the object creator (to maintain and update the object), and republishes them to recipients.

It is expected that that there will be CTI brokers, such as an ISAC, that aggregate, validate and maintain a feed of STIX objects. In these cases, the object(s) may originally come from a member of the ISAC, but the ISAC will want to maintain the object(s) and to update the object(s) based upon feedback from other members or research they themselves did, without involving the member that produced it. In these cases, it is expected that the ISAC will take the original STIX object, and reissue the object w/ a new `identifier`. The ISAC is the object creator of this new object, and will be able update the object. If the ISAC simply republishes the original object from the member, any changes made to the object would have to be issued by the same member that published it. This would impact the ability of the ISAC to get up to date information to it’s members.

Step #	STIX Object	Broker Action

1	{
	"type": "example",
	"id": "example-1",
	"revision":1
	}
		Broker receives this object

2	{
	"type": "example",
	"id": "example-B1",
	"revision": 1
	}
		Broker sends the object, as the object creator, with a new **`id`** and **`revision`** values.

Example: Proxy
In contrast to a broker, a proxy simply passes some or all STIX objects to recipients without modification. An example is an entity that accepts content from a number of object creators and distributes them to all consumers, unmodified.

Note that if a proxy filters the set of objects that might change the picture of the intelligence, and is in effect "updating" the content. But, as they are not updating any of the individual objects, for the purposes of object versioning only they aren't an object updater.

Step #	STIX Object	Proxy Action

1	<pre>{
	"type": "example",
	"id": "example-1",
	"revision": 1
	}</pre>
		Proxy receives this object

2	<pre>{
	"type": "example",
	"id": "example-1",
	"revision": 1
	}</pre>
		Proxy sends the object, as received.

3.3. Confidence

STUB
< add description and details >

Open Questions:

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

3.4. Controlled Vocabularies

	Status: Review **MVP**: Yes

All vocabularies in the CTI specifications are either "*Uncontrolled*" or "*Controlled*". Each vocabulary also supports an extension point to support additional / external vocabularies. The key name for the extension point (of `vocab-ext` type) is **`[vocabulary_field_name]_ext`**. The

vocabulary fields also support the ability to have a fallback value. Using the value of `other` in the main field indicates no fallback value.

Uncontrolled - This type represents a vocabulary that is not yet defined in the specification, but may be defined in a future version. This type means you can you can use any string value you want or use the extension point to specify a value from your own controlled vocabulary.

Controlled - This type represents a vocabulary that is defined in the specification. You can use this controlled vocabulary or use the extension point to specify a value from your own controlled vocabulary.

3.4.1. Examples

In this example the field *cti_type* is an uncontrolled vocabulary, which means you can use any string value you want

```
{
  ...,
  "cti_type": "foo bar"
  ...
}
```

In this example the field *cti_type* is a controlled vocabulary, where you use something from the defined vocabulary in the specification.

```
{
  ...,
  "cti_type": "malware",
  ...
}
```

In this example the field *cti_type* is a controlled vocabulary (example is the same for an uncontrolled vocabulary), however, you want to use your own controlled vocabulary with no fallback / default value. Using the value of `other` in the main field indicates no fallback value.

```
{
  ...,
  "cti_type": "other",
  "cti_type_ext": {
    "value": "malware type foo",
    "vocab": "my name or url to my super cool vocab"
  },
  ...
}
```

In this example the field *cti_type* is a controlled vocabulary (example is the same for an uncontrolled vocabulary), however, you want to use your own controlled vocabulary with a fallback / default value from the defined vocabulary.

```
{
  ...,
  "cti_type": "malware",
  "cti_type_ext": {
     "value": "malware type foo",
     "vocab": "my name or url to my super cool vocab"
  },
  ...
}
```

In this example the field *cti_type* is a controlled vocabulary (not valid for an uncontrolled vocabulary), however, you want to use some arbitrary string value that is not part of the defined vocabulary or any other vocabulary with no fallback / default value. Note the use of the value `other` in this case.

```
{
  ...,
  "cti_type": "other",
  "cti_type_ext": {
     "value": "malware type foo"
  },
  ...
}
```

In this example the field cti_type is a controlled vocabulary (not valid for an uncontrolled vocabulary), however, you want to use some arbitrary string value that is not part of the defined vocabulary or any other vocabulary and you want to add a fallback / default value from the defined vocabulary.

```
{
  ...,
  "cti_type": "malware",
  "cti_type_ext": {
     "value": "malware type foo"
  },
  ...
}
```

3.5. ENUMs

	Status: Concept **MVP**: Undecided

< TODO add description. We need to explain how ENUMs are different from Controlled Vocabularies >

3.6. Vendor Defined Fields

	Status: Draft **MVP**: Yes

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 in this document; these fields are called **Vendor Defined Fields**. This section provides guidance and requirements for how producers can use vendor-defined fields and how consumers should interpret them in order to extend STIX in an interoperable manner.

3.6.1. Processing Rules

- Producers **MAY** create any number of Vendor Defined Fields in a STIX document.
- Vendor Defined Fields **SHOULD** start with "x_" following by a source unique identifier (like a domain name), an underscore and then the name. For example: **`x_examplecom_customfield`**.
- Vendor Defined Fields **MUST** be uniquely named when produced by the same source and **SHOULD** use a consistent namespace prefix (e.g., a domain name).
- Rules for processing Vendor Defined Fields **SHOULD** be well defined and accessible to any consumer that would be reasonably expected to parse them.
- Vendor Defined Fields **SHOULD** only be used when there is no existing field defined by the STIX specification that fulfills that need.

Any consumer that receives a STIX document with one or more Vendor Defined Fields **MAY**:

- process the fields in the manner intended by the producer, if known
- refuse to process the document further
- silently ignore non-understood fields and continue processing the document

Producers of STIX documents that contain Vendor Defined Fields **SHOULD** be well aware of the variability of consumer behavior depending on whether or not the consumer understands the Vendor Defined Fields present in the STIX document. Vendor defined fields 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.

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

Consumers that receive a STIX document that contains one or more Vendor Defined Fields that are understood **MUST** process the Vendor Defined Fields according to the rules for that Vendor Defined Field.

3.6.2. Examples

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

3.7. Data Markings

	Status: Review **MVP**: Yes

Data markings provide the ability for producers to convey to consumers how they may use and share the marked data that they receive.

The CTI Common specification defines two types of data markings:

- **Object-level** data markings define how markings are applied to entire packages and top-level objects
- **Granular** data markings define how markings are applied to one or more individual data points within a package or top-level object

3.7.1. Object-Level Markings

Object-level markings are contained in the **`object_marking_refs`** field, which is an array of ID references (of type `identifier`) that resolve to objects of type `marking-definition`. This field **MUST ONLY** be used on packages and top-level objects: when used at the *package* level, the markings referenced in the list apply to the package itself and to each of the top-level objects in the package. When used on a *top-level object*, the markings apply only to that marked object.

3.7.2. Granular Markings

Granular markings are contained in the **`granular_markings`** field, which is an array of `granular-marking` objects. Each of those objects contains a list of content selectors to select what is marked and a list of marking references that refer to the markings to be applied. The

consumer must evaluate each `granular-marking` in the list individually. For each marking, the list of **`content_selectors`** must be evaluated to determine the set of content that the markings are applied against. The list of `marking-definition` objects referenced in the **`marking_refs`** field are then applied to the selected content.

3.7.3. Precedence Rules

Markings of the same type may override other markings of that type. Processors MUST honor the following order of precedence when encountering more than one marking application with the same type:

- Object markings applied at the package level. Within the **`marking_refs`** field, markings appearing later override markings appearing earlier. These are overridden by,
- Object markings applied at the object level. Within the **`marking_refs`** field, markings appearing later override markings appearing earlier. These are overridden by,
- Granular markings applied at the package level. Within the **`granular_markings`** and **`content_selectors`** fields, markings appearing later override markings type appearing earlier. These are overridden by,
- Granular markings applied at the object level. Within the **`granular_markings`** and **`content_selectors`** fields on the object, markings appearing later override markings appearing earlier.

Markings of different types *never* override each other.

3.7.4. Interoperability

Producers **MAY** create any combination of object-level and granular data markings. Producers **MUST** ensure that all markings they create comply with the functional and data marking requirements defined in this document.

A consumer **MAY** support:

- No data markings (known as a **Level 0 Marking Consumer**)
- **Object-level** data markings (known as a **Level 1 Marking Consumer**)
- Both **object-level** and **granular** data markings (known as a **Level 2 Marking Consumer**)

A **Level 0 Markings Processor** is not able to process data markings. A **Level 1 Markings Processor** is only able to process object-level data markings. A **Level 2 Markings Processor** is able to process both object-level markings and granular markings.

3.7.5. Level 0

- A Level 0 processor who receives a package that contains either the **`object_marking_refs`** field or the **`granular_markings`** field at the package level **MUST** reject the entire package.
- A Level 0 processor who receives a package that **DOES NOT** contain the **`object_marking_refs`** field or the **`granular_markings`** field at the package level **MAY** accept the package.
- A Level 0 processor processing an object that contains the **`object_marking_refs`** field or the **`granular_markings`** field **MUST** reject the object. It **MAY** continue to process the rest of the document.

3.7.6. Level 1

- A Level 1 processor who receives a package that contains the **`granular_markings`** field at the package level **MUST** reject the document.
- A Level 1 processor who receives a document that DOES NOT contain the **`granular_markings`** field at the package level **MAY** accept the document.
- A Level 1 processor processing a document that contains an object with the **`granular_markings`** field **MUST** reject the object. It **MAY** continue to process the rest of the document.
- A Level 1 processor **MUST** process all object-level markings applied in the **`object_marking_refs`** field per the mechanisms outlined in this specification.

3.7.7. Level 2

- **2.1**: A Level 2 processor **MUST** process all object-level markings applied in the **`object_marking_refs`** field and granular markings applied in the **`granular_markings`** field per the mechanisms outlined in this specification.

3.7.8. Resolving References

- Level 1 and Level 2 processors that are unable to resolve a reference to a marking definition **MUST** reject content marked by that definition.

3.7.9. Examples

```
{
  "type": "indicator",
  "id": "indicator--089a6ecb-cc15-43cc-9494-767639779235",
  "spec_version": "stix-2.0",
  "object_marking_refs": ["marking-definition--a82a2ecb-2c1a-42cn-9494-772156779431"],
  "granular_markings": [
    {
      "content_selectors": ["$.description"],
```

```
      "marking_refs": ["marking-definition--089a6ecb-cc15-43cc-9494-767639779123"]
  ],
  "description": ["Some description"]
}

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

4. Core Types

<enter description>

4.1. Identifiers

Type Name: `identifier`	**Status:** Draft **MVP**: Yes

This section defines the `identifier` type. Within STIX and CybOX many objects have identifiers (IDs) that uniquely identify them among all objects. These identifiers and fields that references these identifiers (by having a type of `identifier`) **MUST** conform to the production algorithm and format defined in this section.

4.1.1. Format

```
[construct-type]--[uuid]
```

- **`construct-type`**: Construct type is the exact value from the **`type`** field of the construct being identified or referenced.
- **`uuid`**: UUID is an RFC 4122 compliant Version 4 UUID. This field **MUST** be generated according to the algorithm(s) defined in RFC 4122, Section 4.4 (Version 4 UUID).

4.1.2. Examples

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

4.2. Timestamps

Type Name: `timestamp`	**Status:** Draft **MVP**: Yes

This section defines the `timestamp` type. All discrete timestamps (i.e. not time ranges or relative times) in the CTI specifications are made up of two fields: the timestamp field itself, containing the time, and an optional field that indicates the precision of the timestamp.

4.2.1. Format

```
YYYY-MM-DDTHH:mm:ss.ssssssZ
```

- The timestamp field **MUST** be a valid RFC 3339-formatted timestamp.
- The timestamp **MUST** be represented in the UTC timezone and **MUST** use the 'Z' designation to indicate this.
- The optional precision field, if present, **MUST** be one of "year", "month", "day", "hour", "minute", or "second".
    - The default value for the precision field is "second", so omitting the field is equivalent to explicitly specifying "second".
    - A value of "second" indicates that the value in the timestamp field is precise to the number of digits in the seconds value (including any fractional seconds, such as milliseconds).
    - 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 1300 and the precision value is "hour", the window is 1300-1400.
    - When specifying a precision other than "second", the time portion of the timestamp field **MUST** contain zeroes 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 is always 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”.*

4.2.2. Examples

A timestamp know only to a year would look like:

```
{
  "timestamp": "2016-01-01T00:00:00Z",
  "timestamp_precision": "year"
}
```

A timestamp known only to an hour would look like:

```
{
  "timestamp": "2016-01-20T12:00:00Z",
  "timestamp_precision": "hour"
}
```

A timestamp known to a second would look like:

```
{
  "timestamp": "2016-01-20T12:31:12Z"
}
```

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

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

4.3. Extended Vocabulary

Type Name: `vocab-ext`	**Status:** Review **MVP**: Yes

<enter description>

Property Name	Type	Description
value (required)	`string`	Arbitrary value or value from an alternate vocab.
vocab (optional)	`string`	Name or location of alternate vocab

4.3.1. Examples

```
{
  "type": "indicator",
  "indicator_types": ["other"],
  "indicator_types_ext": [{
    "value": "malware type foo",
    "vocab": "my name or url to my super cool vocab"
```

```
  }]
}

{
  "type": "indicator",
  "indicator_types": ["other"]
  "indicator_types_ext": [{
     "value": "malware type foo"
  }]
}
```

4.4. Granular Markings

Type Name: `granular-marking`	**Status:** Review **MVP**: Undecided

This section defines the `granular-marking` type. Granular markings are used within STIX and CybOX to allow producers to mark individual fields and data points within a package or top-level object.

The consumer must evaluate each selector in the **`content_selectors`** list and apply each marking in the **`marking_refs`** list to that content.

Property Name	Type	Description
`content_selectors` (required)	`array` of type `string`	A list of selectors for content within this object. The markings referenced in the **`content_selectors`** field are applied to the selected content. In the JSON MTI specification, this is a list of JSONPath statements. The context root that these statements must be applied against is the object that contains the `granular-marking` list that this is contained in (i.e. for a package, the package, and for a top-level object, that object).
`marking_refs` (required)	`array` of type `identifier`	The list of markings that apply to the fields selected by **`content_selectors`**.

5. Common Properties

5.1. STIX Core

Type Name: `stix-core`	**Status:** Development **MVP**: Yes

STIX Core is a base set of properties that are inherited by top-level STIX objects.

5.1.1. Properties

Property Name	Type	Description
`type` (required)	`string`	The type of construct being represented. This is a fixed value for each top-level object, formatted as all lowercase with '-' as a separator (e.g. `attack-pattern`). This is a serialization-based field and derives from the type of the construct in the model.
`id` (required)	`identifier`	A globally unique identifier for this object
`spec_version` (required)	`string`	The release version of the language specification used to represent this construct
`created_time` (required)	`timestamp`	The time that the object with this ID was created
`created_by_ref` (optional)	`identifier`	The ID of the Identity Object that describes who created this object (`source--`).
`object_marking_refs` (optional)	`array` of type `identifier`	The list of markings applied to this object
`granular_markings` (optional)	`array` of type `granular-marking`	The set of granular markings applied to this object

5.1.2. Common Relationships

The following descriptions of relationships are defined as part of the CTI Common specification as they apply to all objects within STIX and CybOX.

Kind of Relationship	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.
`derived-from`	*<Object>*	*<Object of same type>*	Allows recording that an Object was derived from a different Object of the same type.
`suggested-update`	*<Object>*	*<Object of same type>*	Allows associating an Object produced by a third-party containing suggested updates with the original Object to which the suggested updates apply.
`other`	*<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.

5.1.3. Examples

This example shows the simultaneous use of a Level 1 marking and a Level 2 marking on the Indicator top-level object. The Level 2 marking on the description takes precedence over the Level 1 marking for the rest of the object.

```
{
  "type": "indicator",
  "id": "indicator--089a6ecb-cc15-43cc-9494-767639779235",
  "spec_version": "stix-2.0",
  "created_time": "2016-01-10T13:15:18Z",
```

```
  "created_by_ref": "source--089a6ecb-cc15-43cc-9494-767639779231",
  "object_marking_refs": ["data-marking--089a6ecb-cc15-43cc-9494-767639779123"],
  "granular_markings": [
    {
      "content_selectors": ["$.description"],
      "marking_refs": ["data-marking--089a6ecb-cc15-43cc-9494-767639779124"]
    }
  ],
  "description": ["Some description"]
}
```

5.2. STIX Descriptive

Type Name: `descriptive-properties`	**Status:** Development **MVP**: Yes

<enter description>

Open Questions:

1. i18n internationalization
2. multiple descriptions for marking
3. structured text, need to address this and see if we want to still do it or just define something like mark-down as the only option.

5.2.1. Properties

Property Name	Type	Description
`title` (optional)	`string`	A human readable title for the construct.
`descriptions` (optional)	`array` of type `string`	A prose description of the construct.

6. Common Types

< enter description >

6.1. Impact

Type Name: `impact`	**Status:** Concept **MVP**: Undecided

Impact is used throughout the STIX specification wherever a construct needs to represent estimated or actual impact as a result of a threat.

Open Questions:

1. Allan has a comment about do we really want or need to do this.. Is this really MVP
2. Need to define a min/max on level and is it similar to confidence (0-100)

Property Name	Type	Description
`level` (optional)	`integer`	The estimated severity of the impact.
`intended_effects` (optional)	`array` of type `intended-effect-cv`	A list of effects that the construct intends to create (e.g. “Fraud”, “Harassment”)
`description` (optional)	`string`	A prose description of the impact
`confidence` (optional)	`< TO DO >`	The confidence in this impact statement, using the STIX default vocabulary..
`confidence_ext` (optional)	`vocab-ext`	The confidence in this impact statement, using a third-party vocabulary.

6.2. Statement (statement)

Type Name: `statement`	**Status:** Concept **MVP**: Undecided

Statements are used to describe analysis assertions within an object. They include an controlled vocabulary field,a description, and a confidence assertion.

Open Questions:

1. Allan Thomson, not sure why this is required in MVP 2.0. Is the intent of this type to just be a general purpose description field? If so, then why not just a vendor specific extension field that allows analysis information to complement the additional intel.

Property Name	Type	Description
`value` (required)	?	The value of this statement from the vocabulary. The vocabulary is set by usages of `statement`.
`value_ext` (optional)	`vocab-ext`	A value in an extended vocabulary
`description` (optional)	`string`	A description of this statement
`confidence` (required)	`< TO DO >`	The credibility of this statement
`confidence_ext` (optional)	`vocab-ext`	The confidence in this statement, using a third-party vocabulary.