# TAXII™ Version 2.1.

## Working Draft 01

## 10 April 2018


**Technical Committee:**

OASIS Cyber Threat Intelligence (CTI) TC

**Chair:**

Richard Struse (rjs@mitre.org), MITRE Corporation

**Editors:**

Bret Jordan (bret_jordan@symantec.com), Symantec Corp.
Drew Varner (drew.varner@ninefx.com), NineFX, Inc.


**Related work:**

This specification replaces or supersedes:

- *TAXII™ Version 2.0.* Edited by John Wunder, Mark Davidson, and Bret Jordan. Latest version: http://docs.oasis-open.org/cti/taxii/v2.0/taxii-v2.0.html.

This specification is related to:

- *STIX™ Version 2.0. Part 1: STIX Core Concepts.* Edited by Bret Jordan, John Wunder, and Rich Piazza. Latest version: *http://docs.oasis-open.org/cti/stix/v2.0/csd01/part1-stix-core/stix-v2.0-csd01-part1-stix-core.html*.
- *STIX™ Version 2.0. Part 2: STIX Objects.* Edited by Bret Jordan, John Wunder, and Rich Piazza. Latest version: *http://docs.oasis-open.org/cti/stix/v2.0/csd01/part2-stix-objects/stix-v2.0-csd01-part2-stix-objects.html*.
- *STIX™ Version 2.0. Part 3: Cyber Observable Core Concepts.* Edited by Ivan Kirillov and Trey Darley. Latest version: *http://docs.oasis-open.org/cti/stix/v2.0/csd01/part3-cyber-observable-core/stix-v2.0-csd01-part3-cyber-observable-core.html*.
- *STIX™ Version 2.0. Part 4: Cyber Observable Objects.* Edited by Ivan Kirillov and Trey Darley. Latest version: *http://docs.oasis-open.org/cti/stix/v2.0/csd01/part4-cyber-observable-objects/stix-v2.0-csd01-part4-cyber-observable-objects.html*.
- *STIX™ Version 2.0. Part 5: STIX Patterning.* Edited by Ivan Kirillov and Trey Darley. Latest version: *http://docs.oasis-open.org/cti/stix/v2.0/csd01/part5-stix-patterning/stix-v2.0-csd01-part5-stix-patterning.html*.


**Abstract:**

TAXII™ is an application layer protocol for the communication of cyber threat information in a simple and scalable manner. This specification defines the TAXII RESTful API and its resources along with the requirements for TAXII Client and Server implementations.

**Status:**

This Working Draft (WD) has been produced by one or more TC Members; it has not yet been voted on by the TC or approved as a Committee Draft (Committee Specification Draft or a Committee Note Draft). The OASIS document Approval Process begins officially with a TC vote to approve a WD as a Committee Draft. A TC may approve a Working Draft, revise it, and re-approve it any number of times as a Committee Draft.

This specification is provided under the Non-Assertion Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/cti/ipr.php).

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

**URI patterns:**

Initial publication URI:
http://docs.oasis-open.org/cti/taxii/v2.1/csprd01/taxii-v2.1-csprd01.docx
Permanent "Latest version" URI:
http://docs.oasis-open.org/cti/taxii/v2.1/taxii-v2.1.docx
(Managed by OASIS TC Administration; please don't modify.)

# Table of Contents

# 1 Introduction

TAXII™ is an application layer protocol for the communication of cyber threat information in a simple and scalable manner. This specification defines the TAXII RESTful API and its resources along with the requirements for TAXII Client and Server implementations.

## 1.1 IPR Policy

This specification is provided under the Non-Assertion Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/cti/ipr.php).

## 1.2 Terminology

The key words "**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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

All text is normative except for examples, the overview (section 1.5), and any text marked non-normative.

## 1.3 Normative References


**[HTTP Auth]** IANA, "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry", March 2017, [Online]. Available: https://www.iana.org/assignments/http-authschemes/http-authschemes.xhtml

**[ISO10646]** "ISO/IEC 10646:2014 Information technology -- Universal Coded Character Set (UCS)", 2014. [Online]. Available: http://standards.iso.org/ittf/PubliclyAvailableStandards/c063182_ISO_IEC_10646_2014.zip

**[RFC0020]** Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, http://www.rfc-editor.org/info/rfc20.

**[RFC2119]** Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, http://www.rfc-editor.org/info/rfc2119.

**[RFC2782]** Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, http://www.rfc-editor.org/info/rfc2782.

**[RFC3339]** Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, http://www.rfc-editor.org/info/rfc3339.

**[RFC4033]** Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, http://www.rfc-editor.org/info/rfc4033.

**[RFC4122]** Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, http://www.rfc-editor.org/info/rfc4122.

**[RFC5246]** Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, http://www.rfc-editor.org/info/rfc5246.

**[RFC5280]** Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, http://www.rfc-editor.org/info/rfc5280.

**[RFC6125]** Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, http://www.rfc-editor.org/info/rfc6125.

**[RFC6818]** Yee, P., "Updates to the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 6818, DOI 10.17487/RFC6818, January 2013, http://www.rfc-editor.org/info/rfc6818.

**[RFC6838]** Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, https://www.rfc-editor.org/info/rfc6838.

**[RFC7230]** Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, http://www.rfc-editor.org/info/rfc7230.

**[RFC7231]** Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, http://www.rfc-editor.org/info/rfc7231.

**[RFC7233]** Fielding, R., Ed., Y. Lafon, Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, 10.17487/RFC7233, June 2014, http://www.rfc-editor.org/info/rfc7233.

**[RFC7235]** Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, http://www.rfc-editor.org/info/rfc7235.

**[RFC7540]** Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, http://www.rfc-editor.org/info/rfc7540.

**[RFC7617]** Reschke, J., "The 'Basic' HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015, http://www.rfc-editor.org/info/rfc7617.

**[RFC7671]** Dukhovni, V. and W. Hardaker, "The DNS-Based Authentication of Named Entities (DANE) Protocol: Updates and Operational Guidance", RFC 7671, DOI 10.17487/RFC7671, October 2015, http://www.rfc-editor.org/info/rfc7671.

**[RFC8174]** Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, https://www.rfc-editor.org/info/rfc8174.

**[RFC8259]** Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, https://www.rfc-editor.org/info/rfc8259.

**[TLS1.3]** E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3 draft-ietf-tls-tls13-24", RFC draft, [Online]. Available: https://tools.ietf.org/html/draft-ietf-tls-tls13-24.

## 1.4 Document Conventions

### 1.4.1 Naming Conventions

All type names, property names and literals are in lowercase. Words in property names are separated with an underscore (_), while words in type names and string enumerations are separated with a hyphen (Unicode hyphen-minus, U+002D, '-'). All type names, property names, object names, and vocabulary terms are between three and 250 characters long.

### 1.4.2 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.
  - resource and type names are in red with a light red background – `collection`
  - property names are in bold style – **`description`**
  - parameter names in URLs are stylized with angled brackets - **`<api-root>`**
  - literals (values) are in blue with a blue background – `complete`
- All examples in this document are expressed in JSON. They are in `Consolas` 9-point font, with straight quotes, black text and a `light grey background`, and 2-space indentation. JSON examples in this document are representations of JSON Objects. They should not be interpreted as string literals. The ordering of object keys is insignificant. Whitespace before or after JSON structural characters in the examples are insignificant [RFC8259].
- Parts of the example may be omitted for conciseness and clarity. These omitted parts are denoted with ellipses (...).

## 1.5 Overview

Trusted Automated Exchange of Intelligence Information (TAXII) is an application layer protocol used to exchange cyber threat intelligence (CTI) over HTTPS. TAXII enables organizations to share CTI by defining an API that aligns with common sharing models. Specifically, TAXII defines two primary services, Collections and Channels, to support a variety of commonly-used sharing models. Collections allow a producer to host a set of CTI data that can be requested by consumers. Channels allow producers to push data to many consumers; and allow consumers to receive data from many producers. Collections and Channels can be organized by grouping them into an API Root to support the needs of a particular trust group or to organize them in some other way. *Note*: This version of the TAXII specification reserves the keywords required for Channels but does **not** specify Channel services. Channels and their services will be defined in a subsequent version of this specification.

TAXII is specifically designed to support the exchange of CTI represented in STIX. As such, the examples and some features in the specification are intended to align with STIX. This does not mean TAXII cannot be used to share data in other formats; it is designed for STIX, but is not limited to STIX.

### 1.5.1 Discovery

This specification defines two discovery methods. The first is a network level discovery that uses a DNS SRV record [RFC2782]. This DNS SRV record can be used to advertise the location of a TAXII Server within a network (e.g., so that TAXII-enabled security infrastructure can automatically locate an organization's internal TAXII Server) or to the general Internet. See section 3.8 for complete information on advertising TAXII Servers in DNS.

The second discovery method is a Discovery Endpoint (this specification uses the term Endpoint to identify a URL and an HTTP method with a defined request and response) that enables authorized clients

to obtain information about a TAXII Server and get a list of API Roots. See section 4.1 for complete information on the Discovery Endpoint.

### 1.5.2 API Roots

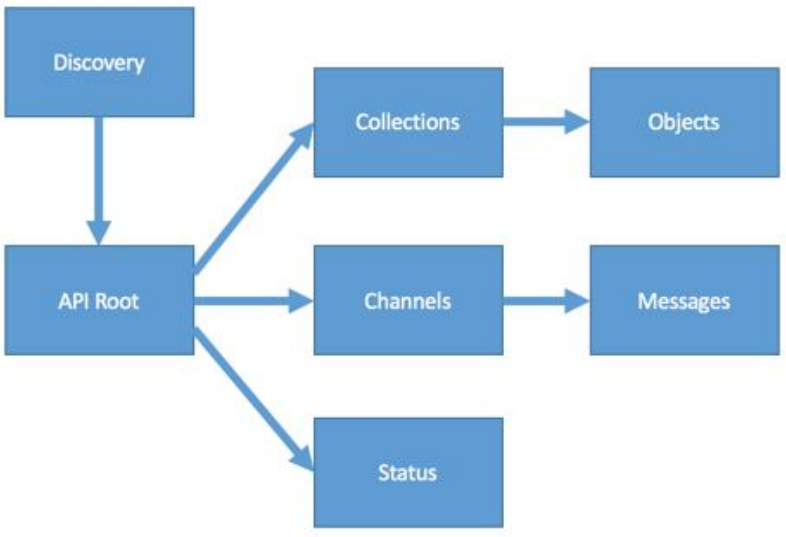
API Roots are logical groupings of TAXII Channels, Collections, and related functionality. A TAXII server instance can support one or more API Roots. API Roots can be thought of as instances of the TAXII API available at different URLs, where each API Root is the "root" URL of that particular instance of the TAXII API. Organizing the Channels and Collections into API Roots allows for a division of content and access control by trust group or any other logical grouping. For example, a single TAXII Server could host multiple API Roots - one API Root for Channels and Collections used by Sharing Group A and another API Root for Channels and Collections used by Sharing Group B.

Each API Root contains a set of Endpoints that a TAXII Client contacts in order to interact with the TAXII Server. This interaction can take several forms:

- Server Discovery, as described above, can be used to learn about the API Roots hosted by a TAXII Server.
- Each API Root might support zero or more Collections. Interactions with Collections include discovering the type of CTI contained in that Collection, pushing new CTI to that Collection, and/or retrieving CTI from that Collection. Each piece of CTI content in a Collection is referred to as an Object.
- Each API Root might host zero or more Channels.
- Each API Root also allows TAXII Clients to check on the Status of certain types of requests to the TAXII Server. For example, if a TAXII Client submitted new CTI, a Status request can allow the Client to check on whether the new CTI was accepted.

Figure 1.1 summarizes the relationships between the components of an API Root.

Figure 1.1



### 1.5.3 Endpoints

An Endpoint consists of a specific URL and HTTP Method on a TAXII Server that a TAXII Client can contact to engage in one specific type of TAXII exchange. For example, each Collection on a TAXII Server has an Endpoint that can be used to get information about it; TAXII Clients can contact the Collection's Endpoint to request a description of that Collection. A separate Endpoint is used for the TAXII Client to collect a manifest of that Collection's Content. Yet another Endpoint is used to get objects from the Collection and, at the same URL, a POST can be used to add objects to the collection. The Endpoints supported by a TAXII Server are summarized in section 3.1 and fully defined in sections 4, 5, and 6.

### 1.5.4 Collections

A TAXII Collection is an interface to a logical repository of CTI objects provided by a TAXII Server and is used by TAXII Clients to send information to the TAXII Server or request information from the TAXII Server. A TAXII Server can host multiple Collections per API Root, and Collections are used to exchange information in a request–response manner.

Figure 1.2 below illustrates how Collection based communications are used when a single TAXII Client makes a request to a TAXII Server and the TAXII Server fulfills that request with information available to the TAXII Server (nominally from a database).

### 1.5.5 Channels

A TAXII Channel is maintained by a TAXII Server and enables TAXII Clients to exchange information with other TAXII Clients in a publish-subscribe model. TAXII Clients can publish messages to Channels and subscribe to Channels to receive published messages. A TAXII Server may host multiple Channels per API Root.

Figure 1.3 below illustrates how Channel communications are used when a single authorized TAXII Client sends a message to the TAXII Server, and that TAXII Server then distributes the message to all authorized TAXII Clients that are connected to the Channel. The arrows in the following diagrams represent data flow.

Figure 1.2

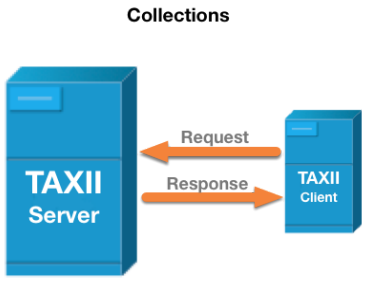
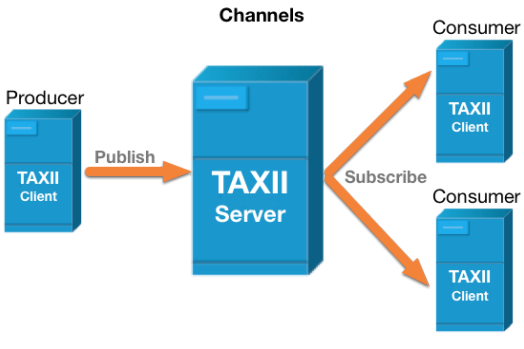



Figure 1.3



### 1.5.6 Transport

The TAXII protocol defined in this specification uses HTTPS (HTTP over TLS) as the transport for all communications.

### 1.5.7 Serialization

This specification uses UTF-8 encoded JSON as defined in [RFC8259] for the serialization of all TAXII resources.

### 1.5.8 Content Negotiation

This specification uses media types (section 3.1.1.1 of [RFC7231]) and an optional "version" parameter in the HTTP Accept header (section 5.3.2 of [RFC7231]) and Content-Type header (section 3.1.1.5 of [RFC7231]) to perform HTTP content negotiation as defined in [RFC7231].

#### 1.5.8.1 Media Types

The STIX and TAXII media types are defined in the following table and are used is both requests and responses.

| Media Type |
| --- |
| `application/taxii+json` |
| `application/stix+json` |

#### 1.5.8.2 Version Parameter

This section defines the optional version parameter that can be used with content negotiation. The version parameter is defined per the guidelines in section 4.3 of [RFC6838]. The value for the version parameter that represents the final version of this specification is "2.1".

**TAXII EDITORS: PLEASE REMOVE THE FOLLOWING TEXT BEFORE CS BALLOT**

While the eventual version indicator for this version of the specification will be "2.1", implementations of draft versions (CSDs) of this specification **SHOULD** instead advertise "2.1-draft01".

This allows pre-final implementations based on Committee Specification Drafts to safely perform content negotiation with each other, even if they would otherwise be incompatible. When this specification is marked as final by the Technical Committee, having advanced to either a CS (committee specification) or an OASIS Standard, implementations **MUST** only advertise "2.1" to represent this specification. Any content that was used prior to this specification becoming final, and has a designation of "-draftXX") **MAY** be converted to the final version or deleted.

| Media Type with Optional Version Parameter | Description |
|---|---|
| `application/taxii+json; version=2.1-draft01` | TAXII version 2.1 CSD01 in JSON |
| `application/stix+json; version=2.1` | STIX version 2.0 in JSON |

### 1.5.9 Authentication and Authorization

Access control to an instance of the TAXII API is specific to the sharing community, vendor, or product and is not defined by this specification.

Authentication and Authorization in TAXII is implemented as defined in [RFC7235], using the `Authorization` and `WWW-Authenticate` HTTP headers respectively.

HTTP Basic authentication, as defined in [RFC7617] is the mandatory to implement authentication scheme in TAXII. As specified in sections 8.2.2 and 8.5.1, TAXII Servers and Clients are required to implement support for HTTP Basic, though other authentication schemes can also be supported. Implementers can allow operators to disable the use of HTTP Basic in their operations.

If the TAXII Server receives a request for any Endpoint that requires authentication, regardless of HTTP method, and either an acceptable `Authorization` header that grants the client access to that object is not sent with the request or the TAXII Server does not determine via alternate means that the client is authorized to access the resource, the TAXII Server responds with a HTTP 401 (Unauthorized) status code and a `WWW-Authenticate` HTTP header.

The `WWW-Authenticate` header contains one or more challenges, which define which authentication schemes are supported by the TAXII Server. The format of the `WWW-Authenticate` HTTP header and any challenges are defined in [RFC7235]. To ensure compatibility, it is recommended that any authentication schemes used in challenges be registered in the IANA Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry [HTTP Auth] .

A TAXII Server may omit objects, information, or optional fields from any response if the authenticated client is not authorized to receive them, so long as that omission does not violate this specification.

### 1.5.10 STIX and Other Content

TAXII is designed with STIX in mind and support for exchanging STIX 2 [*STIX™ Version 2.0. Part 1: STIX Core Concepts*] content is mandatory to implement. Additional content types are permitted, but specific requirements for STIX are present throughout the document. See section 3.6 for more details.

## 1.6 Changes From Earlier Versions

This section lists all of the major changes from the previous 2.0 version of TAXII.

### 1.6.1 TAXII 2.1 Changes and Additions

TAXII 2.1 differs from TAXII 2.0 in the following ways:

1. The discovery URL was changed from /taxii/ to /taxii2/
2. The Manifest Resource was changed to represent individual versions of an object, instead of an object with all of its versions.
3. Item based pagination was removed from this version of the specification.
4. The section on content negotiation was updated.
5. The media types were changed through the document.
6. Clarification was added to say that API Roots can be relative paths as well as absolute paths.
7. Changed version value in API Roots to match media type.

# 2 Data Types

This section defines the names and permitted values of common types used throughout this specification. These types are referenced by the "Type" column in other sections. This table does not, however, define the meaning of any fields using these types. These types may be further restricted elsewhere in the document.

| Type | Description |
|---|---|
| `api-root` | An API Root Resource, see section 4.2.1. |
| `boolean` | A `boolean` is a value of either true or false. Properties with this type MUST have a literal (unquoted) value of `true` or `false`. |
| `bundle` | A STIX Bundle, see section 5 of *STIX™ Version 2.0. Part 1: STIX Core Concepts*. |
| `collection` | A Collection Resource, see section 5.2.1. |
| `collections` | A Collections Resource, see section 5.1.1. |
| `dictionary` | A `dictionary` is a JSON object that captures an arbitrary set of key/value pairs. |
| `discovery` | A Discovery Resource, see section 4.1.1. |
| `error` | An Error Message, see section 3.5.1. |
| `identifier` | An `identifier` 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) [RFC4122]. |
| `integer` | The integer data type represents a whole number. Unless otherwise specified, all integers **MUST** be capable of being represented as a signed 64-bit value. Additional restrictions **MAY** be placed on the type where it is used. |
| `list` | The `list` type defines a sequence of values ordered based on how they appear in the list. The phrasing "`list` of type `<type>`" is used to indicate that all values within the list **MUST** conform to the specified type. For instance, `list` of type `integer` means that all values of the list must be of the `integer` type.<br><br>This specification does not specify the maximum number of allowed values in a `list`, however every instance of a `list` **MUST** have at least one value. Specific TAXII resource properties may define more restrictive upper and/or lower bounds for the length of the list.<br><br>Empty lists are prohibited in TAXII and **MUST NOT** be used as a substitute for omitting optional properties. If the property is required, the list **MUST** be present and **MUST** have at least one value. |

| | |
|---|---|
| | The JSON MTI serialization uses the JSON array type [RFC8259], which is an ordered list of zero or more values. |
| `manifest` | A Manifest Resource, see section 5.6.1. |
| `object` | An Object Resource, see section 3.6. |
| `status` | A Status Resource, see section 4.3.1. |
| `string` | The `string` data type represents a finite-length string of valid characters from the Unicode coded character set [ISO10646] that are encoded in UTF-8. Unicode incorporates ASCII [RFC0020] and the characters of many other international character sets. |
| `timestamp` | The `timestamp` type defines how timestamps are represented in TAXII and is represented in serialization as a `string`.<br><br>• The `timestamp` field **MUST** be a valid RFC 3339-formatted timestamp [RFC3339] using the format `YYYY-MM-DDTHH:mm:ss.[s+]Z` where the "s+" represents 1 or more sub-second values. The brackets denote that sub-second precision is optional, and that if no digits are provided, the decimal place MUST NOT be present.<br>• The timestamp **MUST** be represented in the UTC timezone and **MUST** use the "Z" designation to indicate this. |

# 3 TAXII™ API - Core Concepts

The TAXII API is described as sets of Endpoints. Each Endpoint is identified by the URL that it is accessible at and the HTTP method that is used to make the request. For example, the "Get Collections" Endpoint is requested by issuing a GET to `<api-root>/collections/`. Each Endpoint identifies its URL, which parameters it accepts (including both path parameters and standard parameters), which features it supports (e.g. filtering), and which content types it defines on request and response. It also identifies common error conditions and provides guidance on how to use the Endpoint.

This section defines behavior that applies across Endpoints, such as normative requirements to support each Endpoint, sorting, filtering, and error handling.

## 3.1 Endpoints

Sections 4, 5 and 6 define the set of TAXII Endpoints used in the TAXII API. The following normative requirements apply to each Endpoint:

- The endpoint path in a requests to a TAXII server **MUST** end in a trailing slash "/". For example:
  - A request for a resource without any filter parameters
    /<api-root>/collections/<id>/objects/<object-id>/
  - A request for a resource with some filter parameters.
    /<api-root>/collections/<id>/objects/<object-id>/?match[type]=indicator
- All TAXII requests **MUST** include a media range in the `Accept` header. Requests for TAXII or STIX content **MUST** use the values from section 1.5.8 and **SHOULD** include the optional version parameter.
- All TAXII responses **MUST** include the appropriate media type and version parameter in the `Content-Type` header as defined for that Endpoint.
- TAXII responses **SHOULD** be the highest version of content (e.g., TAXII, STIX) that the server supports if the version parameter in the `Accept` header is omitted during content negotiation.
- TAXII responses with an HTTP success code (200 series) that permit a response body **MUST** include the appropriate response body for the specified content type as identified in the definition of that Endpoint.
- TAXII responses with an HTTP error code (400-series and 500-series status codes, defined by sections 6.5 and 6.6 of [RFC7231]) that permit a response body (i.e. are not in response to a HEAD request) **MUST** contain an `error` message (see section 3.5.1) in the response body.
- Requests with media types in the `Accept` and/or `Content-Type` headers that are defined for that Endpoint **MUST NOT** result in an HTTP 406 (Not Acceptable) or HTTP 415 (Unacceptable Media Type) response.
- Requests with media types in the `Accept` and/or `Content-Type` headers that are not defined for that Endpoint **MAY** be satisfied with the appropriate content or **MAY** result in an HTTP 406 (Not Acceptable) or HTTP 415 (Unacceptable Media Type) response.
- TAXII responses to Endpoints that support filtering **MUST** filter results per the requirements in section 3.4.

The following table provides a summary of the Endpoints (URLs and HTTP Methods) defined by TAXII and the Resources they operate on.

| URL | Methods | Resource Type (section 2) |
|---|---|---|
| **Core Concepts (section 4)** | | |
| /taxii2/ | GET | `discovery` |
| **<api-root>**/ | GET | `api` |
| **<api-root>**/status/**<status-id>**/ | GET | `status` |
| **Collections (section 5)** | | |
| **<api-root>**/collections/ | GET | `collections` |
| **<api-root>**/collections/**<id>**/ | GET | `collection` |
| **<api-root>**/collections/**<id>**/objects/ | GET, POST | `object`* |
| **<api-root>**/collections/**<id>**/objects/**<object-id>**/ | GET | `object`* |
| **<api-root>**/collections/**<id>**/manifest/ | GET | `manifest` |
| **Channels (section 6)** | | |
| <TBD in a future version> | | |

** The actual format of objects is dependent on HTTP Content negotiation, as discussed in section 1.5.8*

## 3.2 HTTP Headers

This section summarizes the HTTP headers and defines custom headers used by this specification.

| Type | Description |
|---|---|
| **Standard Headers** | |
| `Accept` | The Accept header is used by HTTP Requests to specify which Content-Types are acceptable in response. STIX and TAXII define media types and an optional version parameter that can be used in the Accept header. See section 5.3.2 of [RFC7231]. |
| `Authorization` | The Authorization header is used by HTTP Requests to specify authentication credentials. See section 4.2 of [RFC7235]. |
| `Content-Type` | The Content-Type header is used by HTTP to identify the format of HTTP Requests and HTTP Responses. STIX and TAXII define |

| | |
|---|---|
| | media types and an optional version parameter that can be used in the Content-Type header. See section 3.1.1.5 of [RFC7231]. |
| `WWW-Authenticate` | The WWW-Authenticate header is used by HTTP Responses to indicate that authentication is required and to specify the authentication schemes and parameters that are supported. See section 4.1 of [RFC7235]. |
| **Custom Headers** | |
| `X-TAXII-Date-Added-First` | The X-TAXII-Date-Added-First header is an extension header. It indicates the **`date_added`** timestamp of the first object of the response.<br><br>The value of this header **MUST** be a `timestamp`. All HTTP 200 and 206 responses to the following endpoints **MUST** include the `X-TAXII-Date-Added-First` header:<br>● GET **<api-root>**/collections/**<id>**/manifest/<br>● GET **<api-root>**/collections/**<id>**/objects/<br>● GET **<api-root>**/collections/**<id>**/objects/**<object-id>**/<br>Behaviour of this header on any other endpoint, is not defined. |
| `X-TAXII-Date-Added-Last` | The X-TAXII-Date-Added-Last header is an extension header. It indicates the **`date_added`** timestamp of the last object of the response.<br><br>The value of this header **MUST** be a `timestamp`. All HTTP 200 and 206 responses to the following endpoints **MUST** include the `X-TAXII-Date-Added-Last` header:<br>● GET **<api-root>**/collections/**<id>**/manifest/<br>● GET **<api-root>**/collections/**<id>**/objects/<br>● GET **<api-root>**/collections/**<id>**/objects/**<object-id>**/<br>Behaviour of this header on any other endpoint, is not defined. |

## 3.3 Sorting

For Object and Manifest Endpoints, objects returned **MUST** be sorted in ascending order by the date it was added. Meaning, the most recently added object is last in the list.

The Collections Endpoint **MUST** return Collection Resources in a consistent sort order across multiple requests.

## 3.4 Filtering

This section defines the URL query parameters used for matching and filtering content. A TAXII Client can request specific content from a TAXII Server by specifying a set of filters included in the request to

the server. The URL query parameters listed below specifies what to **include** in the response from the TAXII Server. If no URL query parameter is specified then the TAXII Client is requesting that all content be returned for that Endpoint, subject to any default behaviors as listed below.

If any of the URL query parameters are malformed, the TAXII Server **MUST** return an HTTP 400 (Bad Request) status code.

| URL Query Parameters | Description |
|---|---|
| `added_after` | A single timestamp that filters objects to only include those added after the specified timestamp. The value of this parameter is a `timestamp`.<br><br>A request **MUST NOT** have more than one instance of this URL query parameter. If this parameter is provided it **MUST** contain only a single timestamp.<br><br>If no `added_after` URL query parameter is provided, the server **MUST** return the oldest records matching the request first. For example, if a server has 100 records (0-99) and limits requests to 10 records at a time and a client makes a request without an `added_after` URL query parameter, the server would start at record 0 looking for a match and work its way up from oldest to newest finding 10 records that matched the request.<br><br>The `added_after` parameter is not in any way related to dates or times in a STIX object or any other CTI object.<br><br>*Note: The HTTP Date header can be used to identify and correct any time skew between client and server.* |
| `match[<field>]` | The `match` parameter defines filtering on the specified `<field>`. The list of fields that **MUST** be supported is defined per Endpoint as defined in sections 4, 5, and 6. The `match` parameter can be specified any number of times, where each `match` instance specifies an additional filter to be applied to the resulting data and each `<field>` **MUST NOT** occur more than once in a request. Said another way, all match fields are ANDed together.<br><br>All `<field>` parameters are defined in the following table. Requests **MAY** use a `<field>` not defined in this specification, and servers **MAY** ignore fields they do not understand.<br><br>Each field **MAY** contain one or more values. Multiple values are separated by a comma (U+002C COMMA, ",") without any spaces. If multiple values are present, the match is treated as a logical OR. For instance, `?match[type]=incident,malware` specifies a filter for objects that are of type incident OR ttp.<br><br>**Examples**<br>`?match[type]=incident,malware,threat-actor` |

| | |
|---|---|
| | `?match[type]=incident&match[version]=2016-01-01T01:01:01.000Z` |

### 3.4.1 Supported Fields for Match

| Match Field | Description |
|---|---|
| `id` | The identifier of the object(s) that are being requested. When searching for a STIX Object, this is a STIX ID.<br><br>**Examples**<br>`?match[id]=indicator--3600ad1b-fff1-4c98-bcc9-4de3bc2e2ffb`<br>`?match[id]=indicator--3600ad1b-fff1-4c98-bcc9-4de3bc2e2ffb,sighting--4600ad1b-fff1-4c58-bcc9-4de3bc5e2ffd` |
| `type` | The type of the object(s) that are being requested. Only the types listed in this parameter are permitted in the response.<br><br>Requests for types defined in [STIX™ Version 2.0. Part 2: STIX Objects] **MUST NOT** result in an error due to an invalid type.<br><br>Requests for other types not defined in [STIX™ Version 2.0. Part 2: STIX Objects] **MAY** be fulfilled.<br><br>**Examples**<br>`?match[type]=indicator`<br>`?match[type]=indicator,sighting` |
| `version` | The version(s) of the object(s) that are being requested from either an object or manifest endpoint. If no `version` parameter is provided, the server **MUST** return only the latest version for each object matching the remainder of the request.<br><br>Requests **MUST NOT** contain any duplicate version parameters, meaning, each keyword (`all`, `first`, and `last`) and any specific version (`<value>`) **MUST NOT** occur more than once in a request.<br><br>Valid values for the version parameter are:<br>● `last` - requests the latest version of an object. This is the default parameter value if no other version parameter is provided.<br>● `first` - requests the earliest version of an object.<br>● `all` - requests all versions of an object. The `all` keyword **MUST NOT** be used with any other version parameter.<br>● `<value>` - requests a specific version of an object.<br>○ For STIX objects, this requests objects whose modified time matches exactly the provided value and the value **MUST** follow the rules for `timestamp` as defined in [STIX™ Version 2.0. Part 1: STIX Core Concepts]. |

| | |
|---|---|
| | ○ For example: "`2016-01-01T01:01:01.000Z`" tells the server to return the exact STIX object with a modified time of "2016-01-01T01:01:01.000Z".<br>○ For non-STIX objects this value **MAY** be any string that represents the version of that object type. If the target format does not support object versions, this parameter **MUST** be ignored.<br><br>**Examples**<br>`?match[version]=all`<br>`?match[version]=last,first`<br>`?match[version]=first,2018-03-02T01:01:01.123Z,last` |

## 3.5 Errors

TAXII primarily relies on the standard HTTP error semantics (400-series and 500-series status codes, defined by sections 6.5 and 6.6 of [RFC7231]) to allow TAXII Servers to indicate when an error has occurred. For example, an HTTP 404 (Not Found) status code in response to a request to get information about a Collection means that the Collection could not be found. The tables defining the Endpoints in sections 4 and 5 identify common errors and which response should be used, but are not exhaustive and do not describe all possible errors.

In addition to this, TAXII defines an `error` message structure that is provided in the response body when an error status is being returned. It does not, however, define any error codes or error conditions beyond those defined by HTTP.

### 3.5.1 Error Message

**Message Type:** `error`

The `error` message is provided by TAXII Servers in the response body when returning an HTTP error status and contains more information describing the error, including a human-readable **`title`** and **`description`**, an **`error_code`** and **`error_id`**, and a **`details`** structure to capture further structured information about the error. All of the fields are application-specific and clients shouldn't assume consistent meaning across TAXII Servers even if the codes, IDs, or titles are the same.

| Property Name | Type | Description |
|---|---|---|
| **`title`** (required) | `string` | A human readable plain text title for this error. |
| **`description`** (optional) | `string` | A human readable plain text description that gives details about the error or problem that was encountered by the application. |
| **`error_id`** (optional) | `string` | An identifier for this particular error instance. A TAXII Server might choose to assign each error occurrence it's own identifier in order to facilitate debugging. |

| | | |
|---|---|---|
| **error_code** (optional) | string | The error code for this error type. A TAXII Server might choose to assign a common error code to all errors of the same type. Error codes are application-specific and not intended to be meaningful across different TAXII Servers. |
| **http_status** (optional) | string | The HTTP status code applicable to this error. If this property is provided it **MUST** match the HTTP status code found in the HTTP header. |
| **external_details** (optional) | string | A URL that points to additional details. For example, this could be a URL pointing to a knowledge base article describing the error code. Absence of this field indicates that there are no additional details. |
| **details** (optional) | dictionary | The details property captures additional server-specific details about the error. The keys and values are determined by the TAXII Server and **MAY** be any valid JSON object structure. |

**Examples**

```
{
  "title": "Error condition XYZ",
  "description": "This error is caused when the application tries to access data...",
  "error_id": "1234",
  "error_code": "581234",
  "http_status": "409",
  "external_details": "http://example.com/ticketnumber1/errorid-1234",
  "details": {
    "somekey1": "somevalue",
    "somekey2": "some other value"
  }
}
```

## 3.6 Object Resource

**Resource Name:** `object`

This resource type is negotiated based on the media type. This specification does not define any form of content wrapper for objects. Instead, objects are the direct payload of HTTP messages.

When returning STIX 2 content (the Content-Type header contains `application/stix+json; version=2.1`) in a TAXII response, the root object **MUST** be a STIX `bundle` per section 5 of *STIX™ Version 2.0. Part 1: STIX Core Concepts*. For example:

- A single indicator in response to a request for an indicator by ID is enclosed in a `bundle`.
- A list of campaigns returned from a Collection is enclosed in a `bundle`.
- An empty response with no STIX objects results in an empty `bundle`.

Definitions for media types other than STIX can be found in their respective specifications.

**Examples**

```
{
  "type": "bundle",
  ...,
  "objects": [
    {
      "type": "indicator",
      "id": "indicator--252c7c11-daf2-42bd-843b-be65edca9f61",
      ...,
    }
  ]
}
```

## 3.7 Property Names

- All property names and string literals **MUST** be exactly the same, including case, as the names listed in the property tables in this specification.
    - For example, the `discovery` resource has a property called `api_roots` and it must result in the JSON key name "api_roots".
- Properties marked required in the property tables **MUST** be present in the JSON serialization of that resource.

## 3.8 DNS SRV Names

Organizations that choose to implement a DNS SRV record in their DNS server to advertise the location of their TAXII Server MUST use the service name `taxii`.

**Examples**

The following example is for a DNS SRV record advertising a TAXII Server for the domain “example.com” located at taxii-hub-1.example.com:443:

```
_taxii._tcp.example.com. 86400 IN SRV 0 5 443 taxii-hub-1.example.com
```

# 4 TAXII™ API - Server Information

The following table provides a summary of the Server Information Endpoints (URLs and HTTP Methods) defined by TAXII and the Resources they operate on.

| URL | Methods | Resource Type |
|---|---|---|
| /taxii2/ | GET | `discovery` |
| **<api-root>**/ | GET | `api-root` |
| **<api-root>**/status/**<status-id>**/ | GET | `status` |

## 4.1 Server Discovery

This Endpoint provides general information about a TAXII Server, including the advertised API Roots. It's a common entry point for TAXII Clients into the data and services provided by a TAXII Server. For example, clients auto-discovering TAXII Servers via the DNS SRV record defined in section 1.5.1 will be able to automatically retrieve a discovery response for that server by requesting the `/taxii2/` path on that domain.

Discovery API responses **MAY** advertise any TAXII API Root that they have permission to advertise, included those hosted on other servers.

| **`GET /taxii2/`**<br><br>`Get information about the TAXII Server.` | |
|---|---|
| **Response Type** | `discovery` - (`application/taxii+json`) |
| **Responses** | **`200`** - The request was successful<br><br>**`404`** - No discovery information could be found or the requester does not have access to get discovery information.<br><br>**`401, 403`** - The client either needs to authenticate or does not have access to get discovery information |

### 4.1.1 Discovery Resource

**Resource Name:** `discovery`

The `discovery` resource contains information about a TAXII Server, such as a human-readable **`title`**, **`description`**, and **`contact`** information, as well as a list of API Roots that it is advertising. It also has an

indication of which API Root it considers the **`default`**, or the one to use in the absence of other information/user choice.

| Property Name | Type | Description |
|---|---|---|
| **`title`** (required) | `string` | A human readable plain text name used to identify this server. |
| **`description`** (optional) | `string` | A human readable plain text description for this server. |
| **`contact`** (optional) | `string` | The human readable plain text contact information for this server and/or the administrator of this server. |
| **`default`** (optional) | `string` | The default API Root that a TAXII Client **MAY** use. Absence of this field indicates that there is no default API Root. The default API Root **MUST** be an item in **`api_roots`**. |
| **`api_roots`** (optional) | `list` of type `string` | A list of URLs that identify known API Roots. This list **MAY** be filtered on a per-client basis.<br><br>API Root URLs **MUST** be HTTPS absolute URLs or relative URLs. API Root relative URLs **MUST** begin with a single `/` character and **MUST NOT** begin with `//` or '../". API Root URLs **MUST NOT** contain a URL query component.<br><br>Examples - Valid<br>https://taxii.example.com:443/<br>https://someserver.example.net/apiroot1/<br>/someapiroot/<br><br>Examples -Invalid<br>//someserver.example.com/apiroot1<br>../someapiroot/<br>https://foo.edu/bar?baz |

**Examples**

*URLs*

```
https://taxii.example.com:443/taxii2/
https://someserver.example.net/taxii2/
```

*GET Request*

```
GET /taxii2/ HTTP/1.1
Host: example.com
Accept: application/taxii+json; version=2.1
```

*GET Response*

```
HTTP/1.1 200 OK
Content-Type: application/taxii+json; version=2.1

{
```

```
  "title": "Some TAXII Server",
  "description": "This TAXII Server contains a listing of...",
  "contact": "string containing contact information",
  "default": "https://example.com/api2/",
  "api_roots": [
    "https://example.com/api1/",
    "https://example.com/api2/",
    "https://example.net/trustgroup1/"
  ]
}
```

## 4.2 Get API Root Information

This Endpoint provides general information about an API Root, which can be used to help users and clients decide whether and how they want to interact with it. Multiple API Roots **MAY** be hosted on a single TAXII Server. Often, an API Root represents a single trust group.

- Each API Root **MUST** have a unique URL.
- Each API Root **MAY** have different authentication and authorization schemes.

| **GET /<api-root>/**<br><br>Get information about a specific API Root. | |
|---|---|
| **Response Type** | `api-root` - (`application/taxii+json`) |
| **Parameters** | `<api-root>` - the base URL of the API Root containing the Collections |
| **Responses** | **200** - The request was successful<br><br>**404** - No API Root could be found or the requester does not have access to get API Root information.<br><br>**401, 403** - The client either needs to authenticate or does not have access to get API Root information. |

### 4.2.1 API Root Resource

**Resource Name:** `api-root`

The `api-root` resource contains general information about the API Root, such as a human-readable **`title`** and **`description`**, the TAXII **`versions`** it supports, and the maximum size (**`max_content_length`**) of the content body it will accept in a PUT or POST request.

| Property Name | Type | Description |
|---|---|---|
| **`title`** (required) | `string` | A human readable plain text name used to identify this API instance. |

| | | |
|---|---|---|
| **description** (optional) | `string` | A human readable plain text description for this API Root. |
| **versions** (required) | `list` of type `string` | The list of TAXII versions that this API Root is compatible with. The values listed in this property **MUST** match the media types defined in Section 1.5.8.1 and **MUST** include the optional version parameter. A value of `application/taxii+json; version=2.1` **MUST** be included in this list to indicate conformance with this specification. |
| **max_content_length** (required) | `integer` | The maximum size of the request body in octets (8-bit bytes) that the server can support. This applies to requests only and is determined by the server. Requests with total body length values smaller than this value **MUST NOT** result in an HTTP 413 (Request Entity Too Large) response. If for example, the server supported 100 MB of data, the value for this property would be determined by 100*1024*1024 which equals 104,857,600. |

**Examples**

*URLs*

```
https://example.com/api1/
https://example.com/api2/
https://example.org/trustgroup1/
```

*GET Request*

```
GET /api1/ HTTP/1.1
Host: example.com
Accept: application/taxii+json; version=2.1
```

*GET Response*

```
HTTP/1.1 200 OK
Content-Type: application/taxii+json; version=2.1

{
  "title": "Malware Research Group",
  "description": "A trust group setup for malware researchers",
  "versions": ["taxii-2.0"],
  "max_content_length": 104857600
}
```

## 4.3 Get Status

This Endpoint provides information about the status of a previous request. In TAXII 2.1, the only request that can be monitored is one to add objects to a Collection (see section 5.4). It is typically used by TAXII Clients to monitor a POST request that they made in order to take action when it is complete.

TAXII Servers **SHOULD** provide status messages at this Endpoint while the request is in progress until at least 24 hours after it has been marked completed.

| **GET /<api-root>/status/<status-id>/**<br><br>Get status information for a specific status ID. | |
|---|---|
| **Response Type** | status - (application/taxii+json) |
| **Parameters** | <api-root> - the base URL of the API Root containing the Collections<br><status-id> - the identifier of the status message being requested |
| **Responses** | **200** - The request was successful<br><br>**404** - No status could be found or the requester does not have access to get status information.<br><br>**401, 403** - The client either needs to authenticate or does not have access to get status information |

### 4.3.1 Status Resource

**Resource Name:** status

The status resource represents information about a request to add objects to a Collection. It contains information about the status of the request, such as whether or not it's completed (**status**) and the status of individual objects within the request (i.e. whether they are still pending, completed and failed, or completed and succeeded).

The status resource is returned in two places: as a response to the initial request (see section 5.4) and in response to a get status request (see section 4.3), which can be made after the initial request to continuously monitor its status.

The list of objects that are still pending and the list of objects that have been added are both lists of strings containing the identifier of the object (e.g., for STIX objects, their **id**). The list of objects that failed to be added is a simple type so that both the **id** and a message indicating why it failed can be provided.

| Property Name | Type | Description |
|---|---|---|
| **id** (required) | identifier | The identifier of this Status resource. |
| **status** (required) | string | The overall status of a previous POST request where an HTTP 202 (Accept) was returned. The value of this property **MUST** be one of complete or pending. A value of complete indicates that this resource will not be updated further and **MAY** be removed in the future. A status of pending indicates that this resource **MAY** update in the future. |

| | | |
|---|---|---|
| **request_timestamp** (optional) | `timestamp` | The datetime of the request that this status resource is monitoring. |
| **total_count** (required) | `integer` | The total number of objects that were in the request. For a STIX `bundle` this would be the number of objects in the `bundle`. |
| **success_count** (required) | `integer` | The number of objects that were successfully created. |
| **successes** (optional) | `list` of type `string` | A list of object IDs that were successfully processed. For STIX objects the STIX ID **MUST** be used here. For object types that do not have their own identifier, the server **MAY** use any value as the **id**. |
| **failure_count** (required) | `integer` | The number of objects that failed to be created. |
| **failures** (optional) | `list` of type `status-failure` | A list of objects that were not successfully processed. |
| **pending_count** (required) | `integer` | The number of objects that have yet to be processed. |
| **pendings** (optional) | `list` of type `string` | A list of objects for objects that have yet to be processed. For STIX objects the STIX ID **MUST** be used here. For object types that do not have their own identifier, the server **MAY** use any value as the **id**. |

**Type Name:** `status-failure`

This type represents an object that was not added to the Collection. It contains the **id** of the object and a **message** describing why it couldn't be added.

| Property Name | Type | Description |
|---|---|---|
| **id** (required) | `string` | The identifier of the object that failed to be created. For STIX objects the **id** **MUST** be the STIX Object **id**. For object types that do not have their own identifier, the server **MAY** use any value as the **id**. |
| **message** (optional) | `string` | A message indicating why the object failed to be created. |

**Examples**

*URLs*

```
https://example.com/api1/status/2d086da7-4bdc-4f91-900e-d77486753710/
https://example.com/api2/status/88dc8293-827e-44f0-a592-4b5302fbe9d3/
https://example.org/trustgroup1/status/5d26743b-4ade-4b7d-8fea-f68119d4f909/
```

*GET Request*

```
GET /api1/status/2d086da7-4bdc-4f91-900e-d77486753710/ HTTP/1.1
Host: example.com
Accept: application/taxii+json; version=2.1
```

*GET Response*

```
HTTP/1.1 200 OK
Content-Type: application/taxii+json; version=2.1

{
  "id": "2d086da7-4bdc-4f91-900e-d77486753710",
  "status": "pending",
  "request_timestamp": "2016-11-02T12:34:34.12345Z",
  "total_count": 4,
  "success_count": 1,
  "successes": [
    "indicator--c410e480-e42b-47d1-9476-85307c12bcbf"
  ],
  "failure_count": 1,
  "failures": [
    {
      "id": "malware--664fa29d-bf65-4f28-a667-bdb76f29ec98",
      "message": "Unable to process object"
    }
  ],
  "pending_count": 2,
  "pendings": [
    "indicator--252c7c11-daf2-42bd-843b-be65edca9f61",
    "relationship--045585ad-a22f-4333-af33-bfd503a683b5"
  ]
}
```

# 5 TAXII™ API - Collections

A TAXII Collection is a logical grouping of threat intelligence that enables the exchange of information between a TAXII Client and a TAXII Server in a request-response manner. Collections are hosted in the context of an API Root. Each API Root **MAY** have zero or more Collections. As with other TAXII Endpoints, the ability of TAXII Clients to read from and write to Collections can be restricted depending on their permissions level.

This sections defines the TAXII API Collection Endpoints (URLs and methods), valid media types, and responses.

The following table provides a summary of the Endpoints (URLs and HTTP Methods) defined by TAXII and the Resources they operate on.

| URL | Methods | Resource Type |
|---|---|---|
| **<api-root>**/collections/ | GET | `collections` |
| **<api-root>**/collections/**<id>**/ | GET | `collection` |
| **<api-root>**/collections/**<id>**/objects/ | GET, POST | `object` |
| **<api-root>**/collections/**<id>**/objects/**<object-id>**/ | GET | `object` |
| **<api-root>**/collections/**<id>**/manifest/ | GET | `manifest` |

## 5.1 Get Collections

This Endpoint provides information about the Collections hosted under this API Root. This is similar to the response to get a Collection (see section 5.2), but rather than providing information about one Collection it provides information about all of the Collections. Most importantly, it provides the Collection's **id**, which is used to request objects or manifest entries from the Collection.

| **GET /<api-root>/collections/**<br><br>Get information about all collections. | |
|---|---|
| **Response Type** | `collections` - (`application/taxii+json`) |
| **Parameters** | `<api-root>` - the base URL of the API Root containing the Collections |
| **Responses** | **200** - The request was successful<br><br>**404** - The Collections resource does not exist or the client does not have access to the Collections resource. |

| | |
|---|---|
| | **401, 403** - The client either needs to authenticate or does not have access to get Collection information. |

### 5.1.1 Collections Resource

**Resource Name:** `collections`

The `collections` resource is a simple wrapper around a list of `collection` resources.

| Property Name | Type | Description |
|---|---|---|
| **collections** (optional) | `list` of type `collection` | A list of Collections. If there are no Collections in the list, this key **MUST** be omitted and the response is an empty object. The `collection` resource is defined in section 5.2.1. |

**Examples**

*GET Request*

```
GET /api1/collections/ HTTP/1.1
Host: example.com
Accept: application/taxii+json; version=2.1
```

*GET Response*

```
HTTP/1.1 200 OK
Content-Type: application/taxii+json; version=2.1

{
  "collections": [
    {
      "id": "91a7b528-80eb-42ed-a74d-c6fbd5a26116",
      "title": "High Value Indicator Collection",
      "description": "This data collection is for collecting high value IOCs",
      "can_read": true,
      "can_write": false,
      "media_types": [
        "application/stix+json; version=2.1"
      ]
    },
    {
      "id": "52892447-4d7e-4f70-b94d-d7f22742ff63",
      "title": "Indicators from the past 24-hours",
      "description": "This data collection is for collecting current IOCs",
      "can_read": true,
      "can_write": false,
      "media_types": [
        "application/stix+json; version=2.1"
      ]
    }
  ]
}
```

## 5.2 Get a Collection

This Endpoint provides general information about a Collection, which can be used to help users and clients decide whether and how they want to interact with it. For example, it will tell clients what it's called and what permissions they have to it.

| **GET /<api-root>/collections/<id>/**<br><br>Get information about a specific collection. | |
|---|---|
| **Response Type** | `collection` - (`application/taxii+json`) |
| **Parameters** | `<api-root>` - the base URL of the API Root containing the Collection<br>`<id>` - the `identifier` of the Collection being requested |
| **Responses** | **200** - The request was successful<br><br>**404** - The Collection could not be found or the requester does not have access to get Collection information.<br><br>**401, 403** - The client either needs to authenticate or does not have access to get Collection information |

### 5.2.1 Collection Resource

**Resource Name:** `collection`

The `collection` resource contains general information about a Collection, such as its **`id`**, a human-readable **`title`** and **`description`**, an optional list of supported **`media_types`** (representing the media type of objects can be requested from or added to it), and whether the TAXII Client, as authenticated, can get objects from the Collection and/or add objects to it.

| Property Name | Type | Description |
|---|---|---|
| **`id`** (required) | `identifier` | The **`id`** property universally and uniquely identifies this Collection. It is used in the Get Collection Endpoint (see section 5.2) as the **`<id>`** parameter to retrieve the Collection. |
| **`title`** (required) | `string` | A human readable plain text title used to identify this Collection. |
| **`description`** (optional) | `string` | A human readable plain text description for this Collection. |
| **`can_read`** (required) | `boolean` | Indicates if the requester can read (i.e., GET) objects from this Collection. If `true`, users are allowed to access the **Get Objects**, **Get an Object**, or **Get Object** |

| | | |
|---|---|---|
| | | **Manifests** endpoints for this Collection. If `false`, users are not allowed to access these endpoints. |
| **`can_write`** (required) | `boolean` | Indicates if the the requester can write (i.e., POST) objects to this Collection. If `true`, users are allowed to access the **Add Objects** endpoint for this Collection. If `false`, users are not allowed to access this endpoint. |
| **`media_types`** (optional) | `list` of type `string` | A list of supported media types for Objects in this Collection. Absence of this field is equivalent to a single-value list containing `application/stix+json`. This list **MUST** describe all media types that the Collection can store. |

**Examples**

*GET Request*

```
GET /api1/collections/91a7b528-80eb-42ed-a74d-c6fbd5a26116/ HTTP/1.1
Host: example.com
Accept: application/taxii+json; version=2.1
```

*GET Response*

```
HTTP/1.1 200 OK
Content-Type: application/taxii+json; version=2.1

{
  "id": "91a7b528-80eb-42ed-a74d-c6fbd5a26116",
  "title": "High Value Indicator Collection",
  "description": "This data collection is for collecting high value IOCs",
  "can_read": true,
  "can_write": false,
  "media_types": [
    "application/stix+json; version=2.1"
  ]
}
```

## 5.3 Get Objects

This Endpoint retrieves objects from a Collection. Clients can search for objects in the Collection, retrieve all objects in a Collection, or paginate through objects in the Collection.

If the Collection specifies **`can_read`** as `false` for a particular client, this Endpoint **MUST** return an HTTP 401 (Unauthorized), HTTP 403 (Forbidden), or HTTP 404 (Not Found) error.

To support searching the Collection, the Endpoint supports filtering as defined in section 3.4. Clients can provide one or more filter parameters to get objects with a specific ID, of a specific type, or with a specific version. Future versions of TAXII will add more advanced filtering capabilities.

When requesting STIX 2 content, that content will always be delivered in a STIX `bundle` even if there are no STIX `object`s returned or there is only one `object` returned. In these cases the `bundle` will be

empty or only contain one `object`. A `bundle` is returned even when requesting a specific object ID, as there may be multiple versions of that object that are returned. Other content types can be requested by using a different `Accept` header, however the specific representation of other content types is not defined.

| **GET /<api-root>/collections/<id>/objects/**<br><br>Get all objects from a collection. | |
|---|---|
| **Response Type** | `bundle` - (`application/stix+json`) |
| **Parameters** | `<api-root>` - the base URL of the API Root containing the Collection<br>`<id>` - the `identifier` of the Collection from which objects are being requested |
| **Filtering** | **Yes** - `id`, `type`, `version` |
| **Responses** | **200** - The request was successful<br><br>**404** - The Objects resource does not exist or the client does not have access to the Objects resource.<br><br>**401, 403** - The client either needs to authenticate or does not have access to get objects in the Collection. |

**Examples**

GET Request

```
GET /api1/collections/91a7b528-80eb-42ed-a74d-c6fbd5a26116/objects/ HTTP/1.1
Host: example.com
Accept: application/stix+json; version=2.1
```

*GET Response*

```
HTTP/1.1 200 OK
Content-Type: application/stix+json; version=2.1

{
  "type": "bundle",
  ...
  "objects": [
    {
      "type": "indicator",
      ...
    }
  ]
}
```

## 5.4 Add Objects

This Endpoint adds objects to a Collection.

If the Collection specifies `can_write` as `false` for a particular client, this Endpoint **MUST** return an HTTP 401 (Unauthorized), HTTP 403 (Forbidden), or HTTP 404 (Not Found) error.

Successful responses to this Endpoint will contain a `status` resource describing the status of the request. The status resource contains an **`id`**, which can be used to make requests to the get status Endpoint (see section 4.3), a **`status`** flag to indicate whether the request is completed or still being processed, and information about the status of the particular objects in the request.

If the request is marked `pending` in the **`status`** field, the client **SHOULD** periodically poll the get status Endpoint to get an updated status until such a time that the **`status`** property returns a value of `complete`. At that point, the request can be considered complete.

When adding STIX 2 content, clients **MUST** deliver all objects in a STIX `bundle`. Other content types **MAY** be added (if the Collection supports it) by using a different `Content-Type` header, however the specific representation of other content types is not defined.

| **POST /<api-root>/Collections/<id>/objects/**<br>Add a new object to a specific collection. | |
|---|---|
| **Request Type** | `bundle` - (`application/stix+json`) |
| **Response Type** | `status` - (`application/taxii+json`) |
| **Parameters** | `<api-root>` - the base URL of the API Root containing the Collection<br>`<id>` - the `identifier` of the Collection to which objects are being added |
| **Responses** | **202** - The request was successful accepted<br><br>**422** - The object type or version is not supported or could not be processed. This can happen, for example, when sending a version of STIX that this TAXII Server does not support and cannot process, when sending a malformed body, or other unprocessable content.<br><br>**401, 403** - The client either needs to authenticate or does not have access to get Collection information |

**Examples**

*POST Request*

```
POST /api1/collections/91a7b528-80eb-42ed-a74d-c6fbd5a26116/objects/ HTTP/1.1
Host: example.com
Accept: application/taxii+json; version=2.1
Content-Type: application/stix+json; version=2.1

{
  "type": "bundle",
  ...
  "objects": [
    {
      "type": "indicator",
      "id": "indicator--c410e480-e42b-47d1-9476-85307c12bcbf",
      ...
    }
  ]
}
```

*POST Response*

```
HTTP/1.1 202 Accepted
Content-Type: application/taxii+json; version=2.1

{
  "id": "2d086da7-4bdc-4f91-900e-d77486753710",
  "status": "pending",
  "request_timestamp": "2016-11-02T12:34:34.12345Z",
  "total_count": 4,
  "success_count": 1,
  "successes": [
    "indicator--c410e480-e42b-47d1-9476-85307c12bcbf"
  ],
  "failure_count": 0,
  "pending_count": 3
}
```

## 5.5 Get an Object

This Endpoint gets an object from a Collection by its **`id`**. It can be thought of as a search where the `match[id]` parameter is set to the `<object-id>` in the path. For STIX 2 objects, the `<object-id>` **MUST** be the STIX **`id`**.

If the Collection specifies `can_read` as `false` for a particular client, this Endpoint **MUST** return an HTTP 401 (Unauthorized), HTTP 403 (Forbidden), or HTTP 404 (Not Found) error.

To support getting a particular version of an object, this Endpoint supports filtering as defined in section 3.4. The only valid match parameter is `version`.

When requesting STIX 2 content, the content will always be delivered in a STIX `bundle` (even if there is zero or only one object returned, in which case the `bundle` will be empty or only contain one object). It is important to note that even when requesting a single object ID, the response can include multiple

versions of the object, thus requiring the need for a `bundle`. Other content types **MAY** be requested by using a different `Accept` header, however the specific representation of other content types is not defined.

| **GET /<api-root>/collections/<id>/objects/<object-id>/**<br><br>Get a specific object from a collection. | |
|---|---|
| **Response Type** | `bundle` - (`application/stix+json`) |
| **Parameters** | `<api-root>` - the base URL of the API Root containing the Collection<br>`<id>` - the `identifier` of the Collection being requested<br>`<object-id>` - the ID of the object being requested |
| **Filtering** | **Yes** - `version` |
| **Responses** | **200** - The request was successful<br><br>**404** - The object could not be found or the requester does not have access to get the object.<br><br>**401, 403** - The client either needs to authenticate or does not have access to get the object. |

**Examples**

*GET Request*

```
GET
/api1/collections/91a7b528-80eb-42ed-a74d-c6fbd5a26116/object/indicator--252c7c11-daf2-42bd-
843b-be65edca9f61/ HTTP/1.1
Host: example.com
Accept: application/stix+json; version=2.1
```

*GET Response*

```
HTTP/1.1 200 OK
Content-Type: application/stix+json; version=2.1

{
  "type": "bundle",
  ...,
  "objects": [
    {
      "type": "indicator",
      "id": "indicator--252c7c11-daf2-42bd-843b-be65edca9f61",
      ...,
    }
  ]
}
```

## 5.6 Get Object Manifests

This Endpoint retrieves a manifest about the objects in a Collection. It supports filtering identical to the get objects Endpoint (see section 5.3) but rather than returning the object itself it returns metadata about the object. It can be used to retrieve metadata to decide whether it's worth retrieving the actual objects.

If the Collection specifies `can_read` as `false` for a particular client, this Endpoint **MUST** return an HTTP 401 (Unauthorized), HTTP 403 (Forbidden), or HTTP 404 (Not Found) error.

This Endpoint supports filtering, which is applied against the source object rather than the manifest entry for an object. Thus, searching the manifest for a **`type`** of `indicator` will return the manifest entries for objects with a type of `indicator`, even though the manifest doesn't have a **`type`** field.

| **`GET /<api-root>/collections/<id>/manifest/`**<br><br>`Get manifest information about the contents of a specific collection.` | |
|---|---|
| **Response Type** | `manifest` - (`application/taxii+json`) |
| **Parameters** | `<api-root>` - the base URL of the API Root containing the Collection<br>`<id>` - the `identifier` of the Collection being requested |
| **Filtering** | **Yes** - `id`, `type`, `version`<br><br>Filtering is based on properties of the objects that the manifest entries represent. For example, filtering by `type=indicator` will return manifest entries for objects with a type of indicator. |
| **Responses** | **200** - The request was successful<br><br>**`404`** - The Manifest resource does not exist or the client does not have access to the Manifest resource.<br><br>**`401, 403`** - The client either needs to authenticate or does not have access to get manifests for objects in the Collection. |

### 5.6.1 Manifest Resource

**Resource Name:** `manifest`

The `manifest` resource is a simple wrapper around a list of `manifest-entry` items.

| Property Name | Type | Description |
|---|---|---|
| **`objects`** (optional) | `list` of type `manifest-entry` | The list of manifest entries for objects returned by the request. If there are no `manifest-entry` items in the list, this key **MUST** be omitted and the response is an empty object. |

**Type Name:** `manifest-entry`

The `manifest-entry` type captures metadata about a single version of an `object`, indicated by the **`id`** property. The metadata includes information such as when that version of the object was added to the Collection, the version of the object itself, and the media type that this specific version of the object is available in.

| Property Name | Type | Description |
|---|---|---|
| **`id`** (required) | `identifier` | The `identifier` of the object that this manifest entry describes. |
| **`date_added`** (optional) | `timestamp` | The date and time this object was added. |
| **`version`** (optional) | `string` | The version of this object.<br><br>For objects in STIX format, the STIX **`modified`** field is the version. |
| **`media_type`** (optional) | `string` | The media type that this specific version of the object can be requested in. This value **MUST** be one of the media types listed on the `collection` resource. |

**Examples**

*GET Request*

```
GET /api1/collections/91a7b528-80eb-42ed-a74d-c6fbd5a26116/manifest/ HTTP/1.1
Host: example.com
Accept: application/taxii+json; version=2.1
```

*GET Response*

```
HTTP/1.1 200 OK
Content-Type: application/taxii+json; version=2.1

{
  "objects": [
    {
      "id": "indicator--29aba82c-5393-42a8-9edb-6a2cb1df070b",
      "date_added": "2016-11-01T03:04:05Z",
      "version": "2016-11-03T12:30:59.000Z",
      "media_type": "application/stix+json; version=2.1"
    },
    {
      "id": "indicator--ef0b28e1-308c-4a30-8770-9b4851b260a5",
      "date_added": "2016-11-01T10:29:05Z",
      "version": "2016-11-03T12:30:59.000Z",
      "media_type": "application/stix+json; version=2.1"
    }
  ]
}
```

# 6 TAXII™ API - Channels

RESERVED

# 7 Customizing TAXII Resources

This section defines how to extent TAXII in an interoperable manner.

## 7.1 Custom Properties

It is understood that there will be cases where certain information exchanges can be improved by adding properties that are not specified nor reserved in this document; these properties are called **Custom Properties**. This section provides guidance and requirements for how TAXII Servers and Clients should use and interpret Custom Properties in order to extend TAXII in an interoperable manner.

Note: The presence of Custom Properties may introduce variability of behavior depending on whether or not the TAXII Server or Client understands the Custom Properties. A reasonable strategy to minimize unwanted variations in behavior is to provide well defined and consistent rules for processing Custom Properties to any TAXII Server or Client that would be reasonably expected to parse them.

### 7.1.1 Requirements

- A TAXII resource **MAY** have any number of Custom Properties.
- Custom Property names **MUST** be in ASCII and are limited to characters a-z (lowercase ASCII) and underscore (_).
- Custom Property names **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 names **SHOULD** be no longer than 30 ASCII characters in length.
- Custom Property names **MUST** have a minimum length of 3 ASCII characters.
- Custom Property names **MUST** be no longer than 256 ASCII characters in length.
- Custom Property names 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 Property names **SHOULD** be unique 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 are no existing properties defined by the TAXII specification that fulfill that need.

TAXII Servers that receive a TAXII Resource with one or more Custom Properties it does not understand **MAY** respond in one of two ways:

1. Either refuse to process the content further and respond to the message with an HTTP 422 (Unprocessable Entity) status code,
2. or silently ignore non-understood properties and continue processing the message.

TAXII Clients that receive a TAXII Resource with one or more Custom Properties it does not understand **MAY** silently ignore non-understood properties and continue processing the message.

The reporting and logging of errors originating from the processing of Custom Properties depends on the TAXII Server and Client implementations and is therefore not covered in this specification.

### Examples

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

# 8 Conformance

## 8.1 TAXII™ Servers

This section describes the types of TAXII Servers that can be implemented and which normative requirements those types of servers must conform to.

### 8.1.1 TAXII™ 2.1 Server

A "TAXII 2.1 Server" is any software that conforms to the following normative requirements:

1. It **MUST** support all requirements for a TAXII Collections Server as defined in section 8.1.2.

### 8.1.2 TAXII™ 2.1 Collections Server

A "TAXII 2.1 Collections Server" is any software that conforms to the following normative requirements:

1. It **MUST** support all requirements as defined in section 3, section 4 and section 5.
2. It **MUST** include all required properties within TAXII Resources, as defined in section 4 and section 5.
3. It **MUST** support all features listed in section 8.2, Mandatory Server Features.
4. It **MAY** support any features listed in section 8.3, Optional Server Features. Software supporting an optional feature **MUST** comply with the normative requirements of that feature.

### 8.1.3 TAXII™ 2.1 Channels Server

RESERVED

## 8.2 Mandatory Server Features

This sections defines the mandatory features that all TAXII Servers must implement.

### 8.2.1 TAXII Server Core Requirements

1. It **MUST** define the URL of the Discovery API to be /taxii2/ and it **MUST** be located at the root of the server, e.g., https://example.com/taxii2/
2. It **MUST** support at least one API Root.
3. It **MAY** support multiple API Roots.
4. It **MAY** implement other HTTP Methods, Content Types, and/or URLs beyond those defined in this specification.
5. It **MUST** be capable of sending HTTP responses for features that it supports whose content is valid TAXII as defined in sections 3, 4, 5, and 6 or STIX as defined in [*STIX™ Version 2.0. Part 1: STIX Core Concepts*].
6. All properties **MUST** conform to the data type and normative requirements for that property.

### 8.2.2 HTTPS and Authentication Server Requirements

1. It **MUST** accept TAXII 2.1 requests using HTTPS [RFC7230].

2. It **MUST** accept connections using TLS version 1.2 [RFC5246] and **SHOULD** accept connections using TLS version 1.3 [TLS1.3] or higher.
3. It **MUST NOT** use the 0-RTT feature of TLS 1.3 [TLS1.3].
4. It **SHOULD NOT** accept any TLS 1.2 connections that use any of the cipher suites that are listed in the cipher suite blacklist in Appendix A of [RFC7540].
5. It **MUST** implement the HTTP Basic authentication scheme per [RFC 7617].
6. It **MAY** permit configurations that enable and/or disable all authentication schemes, including HTTP Basic authentication.
7. It **MAY** implement additional authentication and authorization schemes beyond HTTP Basic, see section 1.5.9.
8. It **MAY** restrict access to clients by omitting specific objects, information, or optional fields from any TAXII response.
9. It **MAY** permit operators to disable all authentication.
10. It **MAY** choose to not respond to (a.k.a. silently ignore) unauthorized requests.

## 8.3 Optional Server Features

This sections defines the optional features that a TAXII Server **MAY** implement.

### 8.3.1 Client Certificate Verification

TAXII 2.1 servers **MAY** choose to verify a client's certificate and use it for authentication. TAXII Servers supporting client certificate verification and authentication **MUST** follow the normative requirements listed in this section.

- The default strategy for TAXII Servers authenticating and verifying certificates **SHOULD** be PKIX as defined in [RFC5280], [RFC6818], [RFC6125] et al.
- It **MAY** support other certificate verification policies such as Certificate Pinning.

## 8.4 TAXII™ Clients

This section describes the types of TAXII Clients that can be implemented and which normative requirements those types of clients must conform to.

### 8.4.1 TAXII™ 2.1 Client

A "TAXII 2.1 Client" is any software that conforms to the following normative requirements:

1. It **MUST** support all requirements for a TAXII Collections Client as defined in section 8.4.2.

### 8.4.2 TAXII™ 2.1 Collections Client

A "TAXII 2.1 Collections Client" is any software that exchanges CTI data with a TAXII 2.1 Collections Server or a TAXII 2.1 Server. A TAXII 2.1 Collections Client conforms to the following normative requirements:

1. It **SHOULD** be capable of looking up and using the TAXII SRV record from DNS.
2. It **MUST** support parsing all properties for resources defined in section 4 and section 5.
3. It **MUST** support all features listed in section 8.5, Mandatory Client Features.

### 8.4.3 TAXII™ 2.1 Channels Client

RESERVED

## 8.5 Mandatory Client Features

This section defines the mandatory features that all TAXII Clients **MUST** support.

### 8.5.1 HTTPS and Authentication Client Requirements

1. It **MUST** initiate TAXII 2.1 requests to a TAXII 2.1 Server using HTTPS [RFC7230].
2. It **MUST** support TLS 1.2 and **SHOULD** use TLS version 1.3 [TLS1.3] or higher
3. It **SHOULD NOT** use TLS 1.2 with any of the cipher suites that are listed in the cipher suite blacklist in Appendix A of [RFC7540].
4. It **MUST** implement the HTTP Basic authentication scheme as a client per [RFC 7617].
5. It **MAY** implement additional authentication and authorization schemes beyond HTTP Basic, see section 1.5.9.

### 8.5.2 Server Certificate Verification

- The default strategy for TAXII Clients authenticating and verifying the server's TLS certificate **SHOULD** be PKIX as defined in [RFC5280], [RFC6818], [RFC6125] et al.
- TAXII Clients **MAY** support other certification verification policies such as:
    - Certificate Pinning: A single or limited set of either hard-coded or physically distributed pinned certificate authorities or end-entity certificates.
    - DANE: DNS-based Authentication of Named Entities [RFC7671]. Systems implementing DANE **SHOULD** also implement DNSSEC [RFC4033].
    - Note that Self-Signed Certificates (like other certificates which cannot be verified by PKIX) **MAY** be supported via Certificate Pinning and/or DANE as noted above.

# Appendix A. Glossary

**API Root** - A grouping of TAXII Channels, Collections, and related functionality.
**Channel** - A publish-subscribe communications method where messages are exchanged.
**CTI** - Cyber Threat Intelligence
**Collection** - A logical group of CTI objects.
**Endpoint** - A combination of a URL and HTTP method with defined behavior in TAXII.
**STIX** - Structured Threat Information Expression (STIX™) is a language and serialization format used to exchange cyber threat intelligence (CTI).
**STIX Content** - STIX documents, including STIX Objects, grouped as STIX Bundles.
**STIX Object** - A STIX Domain Object (SDO) or STIX Relationship Object (SRO).
**TAXII** - Trusted Automated eXchange of Intelligence Information (TAXII™) is an application layer protocol for the communication of cyber threat intelligence (CTI).
**TAXII Client** - A software package that connects to a TAXII Server and supports the exchange of CTI.
**TAXII Server** - A software package that supports the exchange of CTI.

# Appendix B. Acknowledgments

**TAXII Subcommittee Chairs:**

Bret Jordan, Symantec Corp.
Mark Davidson, NC4

**Special Thanks:**

Substantial contributions to this specification from the following individuals are gratefully acknowledged:

Terry MacDonald, Cosive
Jane Ginn, Cyber Threat Intelligence Network, Inc. (CTIN)
Richard Struse, DHS Office of Cybersecurity and Communications
Sergey Polzunov, EclecticIQ
Iain Brown, GDS
Eric Burger, Georgetown University
Jason Keirstead, IBM
Allan Thomson, LookingGlass Cyber
Rich Piazza, MITRE Corporation
Charles Schmidt, MITRE Corporation
John Wunder, MITRE Corporation
Mark Davidson, NC4
John-Mark Gurney, New Context Services, Inc.
Drew Varner, NineFX, Inc.
Dave Cridland, Surevine
Bret Jordan, Symantec Corp.

**Participants:**

The following individuals were members of the OASIS CTI Technical Committee during the creation of this specification and their contributions are gratefully acknowledged:

Robert Coderre, Accenture
Kyle Maxwell, Accenture
David Crawford, Aetna
Marcos Orallo, Airbus Group SAS
Roman Fiedler, AIT Austrian Institute of Technology
Florian Skopik, AIT Austrian Institute of Technology
Ryan Clough, Anomali
Nicholas Hayden, Anomali
Wei Huang, Anomali
Angela Nichols, Anomali
Hugh Njemanze, Anomali
Katie Pelusi, Anomali
Dean Thompson, Australia and New Zealand Banking Group (ANZ Bank)
Alexander Foley, Bank of America
Radu Marian, Bank of America
Sounil Yu, Bank of America

Vicky Laurens, Bank of Montreal
Alexandre Dulaunoy, CIRCL
Andras Iklody, CIRCL
Christian Studer, CIRCL
Raphaïl Vinot, CIRCL
Sarah Kelley, CIS
Syam Appala, Cisco Systems
Ted Bedwell, Cisco Systems
David McGrew, Cisco Systems
Mark-David McLaughlin, Cisco Systems
Pavan Reddy, Cisco Systems
Omar Santos, Cisco Systems
Sam Taghavi Zargar, Cisco Systems
Jyoti Verma, Cisco Systems
Jart Armin, Cyber Threat Intelligence Network, Inc. (CTIN)
Doug DePeppe, Cyber Threat Intelligence Network, Inc. (CTIN)
Jane Ginn, Cyber Threat Intelligence Network, Inc. (CTIN)
Ben Ottoman, Cyber Threat Intelligence Network, Inc. (CTIN)
David Powell, Cyber Threat Intelligence Network, Inc. (CTIN)
Andreas Sfakianakis, Cyber Threat Intelligence Network, Inc. (CTIN)
Andrew Byrne, Dell
Jeff Odom, Dell
Sreejith Padmajadevi, Dell
Ravi Sharda, Dell
Will Urbanski, Dell
Evette Maynard-Noel, DHS Office of Cybersecurity and Communications (CS&C)
Sean Sobieraj, DHS Office of Cybersecurity and Communications (CS&C)
Marlon Taylor, DHS Office of Cybersecurity and Communications (CS&C)
Preston Werntz, DHS Office of Cybersecurity and Communications (CS&C)
Wouter Bolsterlee, EclecticIQ
Adam Bradbury, EclecticIQ
Marko Dragoljevic, EclecticIQ
Oliver Gheorghe, EclecticIQ
Joep Gommers, EclecticIQ
Christopher O'Brien, EclecticIQ
Sergey Polzunov, EclecticIQ
Rutger Prins, EclecticIQ
Andrei Sîrghi, EclecticIQ
Raymon van der Velde, EclecticIQ
Tom Vaughan, EclecticIQ
Ben Sooter, Electric Power Research Institute (EPRI)
Chris Ricard, Financial Services Information Sharing and Analysis Center (FS-ISAC)
Sean Barnum, FireEye, Inc.
Phillip Boles, FireEye, Inc.
Prasad Gaikwad, FireEye, Inc.
Will Green, FireEye, Inc.
Rajeev Jha, FireEye, Inc.
Anuj Kumar, FireEye, Inc.

James Meck, FireEye, Inc.
Shyamal Pandya, FireEye, Inc.
Paul Patrick, FireEye, Inc.
Scott Shreve, FireEye, Inc.
Jon Warren, FireEye, Inc.
Remko Weterings, FireEye, Inc.
Tim Jones, ForeScout
Gavin Chow, Fortinet Inc.
Steve Fossen, Fortinet Inc.
Kenichi Terashita, Fortinet Inc.
Ryusuke Masuoka, Fujitsu Limited
Daisuke Murabayashi, Fujitsu Limited
Derek Northrope, Fujitsu Limited
Toshitaka Satomi, Fujitsu Limited
Koji Yamada, Fujitsu Limited
Kunihiko Yoshimura, Fujitsu Limited
David Lemire, G2
Jonathan Algar, GDS
Iain Brown, GDS
Adam Cooper, GDS
Mike McLellan, GDS
Tyrone Nembhard, GDS
Chris O'Brien, GDS
James Penman, GDS
Howard Staple, GDS
Chris Taylor, GDS
Laurie Thomson, GDS
Alastair Treharne, GDS
Julian White, GDS
Bethany Yates, GDS
Robert van Engelen, Genivia
Eric Burger, Georgetown University
Allison Miller, Google Inc.
Mark Risher, Google Inc.
Yoshihide Kawada, Hitachi, Ltd.
Jun Nakanishi, Hitachi, Ltd.
Kazuo Noguchi, Hitachi, Ltd.
Akihito Sawada, Hitachi, Ltd.
Yutaka Takami, Hitachi, Ltd.
Masato Terada, Hitachi, Ltd.
Adrian Bishop, Huntsman Security
Eldan Ben-Haim, IBM
Allen Hadden, IBM
Sandra Hernandez, IBM
Jason Keirstead, IBM
Chenta Lee, IBM
John Morris, IBM
Devesh Parekh, IBM

Laura Rusu, IBM
Ron Williams, IBM
Paul Martini, iboss, Inc.
Vasileios Mavroeidis, IFI
Jerome Athias, Individual
Joerg Eschweiler, Individual
Stefan Hagen, Individual
Elysa Jones, Individual
Terry MacDonald, Individual
Alex Pinto, Individual
Tim Casey, Intel Corporation
Julie Modlin, Johns Hopkins University Applied Physics Laboratory
Mark Moss, Johns Hopkins University Applied Physics Laboratory
Mark Munoz, Johns Hopkins University Applied Physics Laboratory
Nathan Reller, Johns Hopkins University Applied Physics Laboratory
Pamela Smith, Johns Hopkins University Applied Physics Laboratory
Subodh Kumar, JPMorgan Chase Bank, N.A.
David Laurance, JPMorgan Chase Bank, N.A.
Russell Culpepper, Kaiser Permanente
Beth Pumo, Kaiser Permanente
Michael Slavick, Kaiser Permanente
Gus Creedon, Logistics Management Institute
Wesley Brown, LookingGlass
Jamison Day, LookingGlass
Dennis Hostetler, LookingGlass
Himanshu Kesar, LookingGlass
Allan Thomson, LookingGlass
Ian Truslove, LookingGlass
Chris Wood, LookingGlass
Kent Landfield, McAfee
Greg Back, Mitre Corporation
Jonathan Baker, Mitre Corporation
Desiree Beck, Mitre Corporation
Michael Chisholm, Mitre Corporation
Sam Cornwell, Mitre Corporation
Ivan Kirillov, Mitre Corporation
Michael Kouremetis, Mitre Corporation
Chris Lenk, Mitre Corporation
Nicole Parrish, Mitre Corporation
Richard Piazza, Mitre Corporation
Larry Rodrigues, Mitre Corporation
Jon Salwen, Mitre Corporation
Charles Schmidt, Mitre Corporation
Richard Struse, Mitre Corporation
Alex Tweed, Mitre Corporation
Emmanuelle Vargas-Gonzalez, Mitre Corporation
John Wunder, Mitre Corporation
James Cabral, MTG Management Consultants, LLC.

Scott Algeier, National Council of ISACs (NCI)
Denise Anderson, National Council of ISACs (NCI)
Josh Poster, National Council of ISACs (NCI)
Mike Boyle, National Security Agency
Joe Brule, National Security Agency
Jessica Fitzgerald-McKay, National Security Agency
David Kemp, National Security Agency
Shaun McCullough, National Security Agency
Jason Romano, National Security Agency
John Anderson, NC4
Michael Butt, NC4
Mark Davidson, NC4
Daniel Dye, NC4
Michael Pepin, NC4
Natalie Suarez, NC4
Benjamin Yates, NC4
Sarah Brown, NCI Agency
Oscar Serrano, NCI Agency
Daichi Hasumi, NEC Corporation
Takahiro Kakumaru, NEC Corporation
Lauri Korts-P_rn, NEC Corporation
Trey Darley, New Context Services, Inc.
John-Mark Gurney, New Context Services, Inc.
Christian Hunt, New Context Services, Inc.
Danny Purcell, New Context Services, Inc.
Daniel Riedel, New Context Services, Inc.
Andrew Storms, New Context Services, Inc.
Drew Varner, NineFX, Inc.
Stephen Banghart, NIST
David Darnell, North American Energy Standards Board
James Crossland, Northrop Grumman
Robert Van Dyk, Northrop Grumman
Cheolho Lee, NSRI
Cory Casanave, Object Management Group
Vishaal Hariprasad, Palo Alto Networks
Aharon Chernin, Perch
Dave Eilken, Perch
Philip Royer, Phantom
Sourabh Satish, Phantom
John Tolbert, Queralt Inc.
Jay Heidecker, Seekintoo
Joseph Brand, Semper Fortis Solutions
Duncan Sparrell, sFractal Consulting LLC
Thomas Schreck, Siemens AG
Rob Roel, Southern California Edison
Armen Tashjian, Southern California Edison
Dave Cridland, Surevine Ltd.
Bret Jordan, Symantec Corp.

Robert Keith, Symantec Corp.
Curtis Kostrosky, Symantec Corp.
Chris Larsen, Symantec Corp.
Michael Mauch, Symantec Corp.
Aubrey Merchant, Symantec Corp.
Efrain Ortiz, Symantec Corp.
Mingliang Pei, Symantec Corp.
Kenneth Schneider, Symantec Corp.
Arnaud Taddei, Symantec Corp.
Brian Witten, Symantec Corp.
Juha Haaga, Synopsys
Greg Reaume, TELUS
Alan Steer, TELUS
Crystal Hayes, The Boeing Company
Andrew Gidwani, ThreatConnect, Inc.
Cole Iliff, ThreatConnect, Inc.
Andrew Pendergast, ThreatConnect, Inc.
Jason Spies, ThreatConnect, Inc.
Ryan Trost, ThreatQuotient, Inc.
Nir Yosha, ThreatQuotient, Inc.
Patrick Coughlin, TruSTAR Technology
Chris Roblee, TruSTAR Technology
Mark Angel, U.S. Bank
Brian Fay, U.S. Bank
Joseph Frazier, U.S. Bank
Mark Heidrick, U.S. Bank
Richard Shok, U.S. Bank
Ehab Al-Shaer, UNCC
Bill Chu, UNCC
James Bohling, US Department of Defense (DoD)
Eoghan Casey, US Department of Defense (DoD)
Gary Katz, US Department of Defense (DoD)
Jeffrey Mates, US Department of Defense (DoD)
Evette Maynard-Noel, US Department of Homeland Security
Eric Osterweil, VeriSign
Lee Chieffalo, Viasat
Wilson Figueroa, Viasat
Andrew May, Viasat
Ales Cernivec, XLAB
Anthony Rutkowski, Yanna Technologies LLC

# Appendix C. Revision History

| Revision | Date | Editor(s) | Changes Made |
|---|---|---|---|
| 01 | 2018-04-10 | Bret Jordan<br>Drew Varner | Initial Version |