# TAXII™ Version 2.1

## Committee Specification Draft 01

## 28 September 2018

### Specification URIs

**This version:**

https://docs.oasis-open.org/cti/taxii/v2.1/csd01/taxii-v2.1-csd01.docx (Authoritative)
https://docs.oasis-open.org/cti/taxii/v2.1/csd01/taxii-v2.1-csd01.html
https://docs.oasis-open.org/cti/taxii/v2.1/csd01/taxii-v2.1-csd01.pdf

**Previous version:**

N/A

**Latest version:**

https://docs.oasis-open.org/cti/taxii/v2.1/taxii-v2.1.docx (Authoritative)
https://docs.oasis-open.org/cti/taxii/v2.1/taxii-v2.1.html
https://docs.oasis-open.org/cti/taxii/v2.1/taxii-v2.1.pdf

**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), Individual Member

**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.
- *TAXII™ Version 1.1.1. Part 1: Overview*. Edited by Mark Davidson, Charles Schmidt, and Bret Jordan. Latest version: http://docs.oasis-open.org/cti/taxii/v1.1.1/taxii-v1.1.1-part1-overview.html.

This specification is related to:

- *STIX™ Version 2.0. Part 1: STIX Core Concepts*. Edited by Rich Piazza, John Wunder, and Bret Jordan. Latest version: http://docs.oasis-open.org/cti/stix/v2.0/stix-v2.0-part1-stix-core.html.
- *STIX™ Version 2.0. Part 2: STIX Objects*. Edited by Rich Piazza, John Wunder, and Bret Jordan. Latest version: http://docs.oasis-open.org/cti/stix/v2.0/stix-v2.0-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/stix-v2.0-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/stix-v2.0-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/stix-v2.0-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 document was last revised or approved by the OASIS Cyber Threat Intelligence (CTI) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=cti#technical.

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

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.

**Citation format:**

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

**[TAXII-v2.1]**

*TAXII™ Version 2.1*. Edited by Bret Jordan and Drew Varner. 28 September 2018. OASIS Committee Specification Draft 01. https://docs.oasis-open.org/cti/taxii/v2.1/csd01/taxii-v2.1-csd01.html. Latest version: https://docs.oasis-open.org/cti/taxii/v2.1/taxii-v2.1.html.

# Notices



All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

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

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

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

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



STIX™, CYBOX™, AND TAXII™ (STANDARD OR STANDARDS) AND THEIR COMPONENT PARTS ARE PROVIDED "AS IS" WITHOUT ANY WARRANTY OF ANY KIND, EITHER EXPRESSED, IMPLIED, OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY THAT THESE STANDARDS OR ANY OF THEIR COMPONENT PARTS WILL CONFORM TO SPECIFICATIONS, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR FREEDOM FROM INFRINGEMENT, ANY WARRANTY THAT THE STANDARDS OR THEIR COMPONENT PARTS

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

# Table of Contents

# 1 Introduction

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 1.6), 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 Non-Normative References

**[RFC1738]** Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, DOI 10.17487/RFC1738, December 1994, https://www.rfc-editor.org/info/rfc1738.

**[RFC6545]** Moriarty, K., "Real-time Inter-network Defense (RID)", RFC 6545, DOI 10.17487/RFC6545, April 2012, https://www.rfc-editor.org/info/rfc6545.

**[RFC6749]** Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, https://www.rfc-editor.org/info/rfc6749.

**[RFC7486]** Farrell, S., Hoffman, P., and M. Thomas, "HTTP Origin-Bound Authentication (HOBA)", RFC 7486, DOI 10.17487/RFC7486, March 2015, https://www.rfc-editor.org/info/rfc7486.

**[RFC7804]** Melnikov, A., "Salted Challenge Response HTTP Authentication Mechanism", RFC 7804, DOI 10.17487/RFC7804, March 2016, https://www.rfc-editor.org/info/rfc7804.

**[RFC7515]** Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, https://www.rfc-editor.org/info/rfc7515.

**[RFC7516]** Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, https://www.rfc-editor.org/info/rfc7516.

**[RFC7616]** Shekh-Yusef, R., Ed., Ahrens, D., and S. Bremer, "HTTP Digest Access Authentication", RFC 7616, DOI 10.17487/RFC7616, September 2015, https://www.rfc-editor.org/info/rfc7616.

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

**[RFC7797]** Jones, M., "JSON Web Signature (JWS) Unencoded Payload Option", RFC 7797, DOI 10.17487/RFC7797, February 2016, https://www.rfc-editor.org/info/rfc7797.

**[RFC8322]** Field, J., Banghart, S., and D. Waltermire, "Resource-Oriented Lightweight Information Exchange (ROLIE)", RFC 8322, DOI 10.17487/RFC8322, February 2018, https://www.rfc-editor.org/info/rfc8322.

**[IANA AUTH]** IANA, "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry", February 2014, https://www.iana.org/assignments/http-authschemes/http-authschemes.xhtml.

**[NIST RBAC]** NIST Computer Security Resource Center (CSRC), "Role Based Access Control (RBAC)", November 2016, https://csrc.nist.gov/projects/role-based-access-control.

**[UNICODE]** Davis, M. and Suignard, M., "UNICODE Security Considerations", Unicode Technical Report #36, September 2014, https://www.rfc-editor.org/info/rfc8322.

## 1.5 Document Conventions

### 1.5.1 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.

### 1.5.2 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 `Consolas` 9-point font, with straight quotes, black text, 2-space indentation, and sometimes in a `light grey background`. 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 (...).

## 1.6 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.

### 1.6.1 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.

### 1.6.2 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.

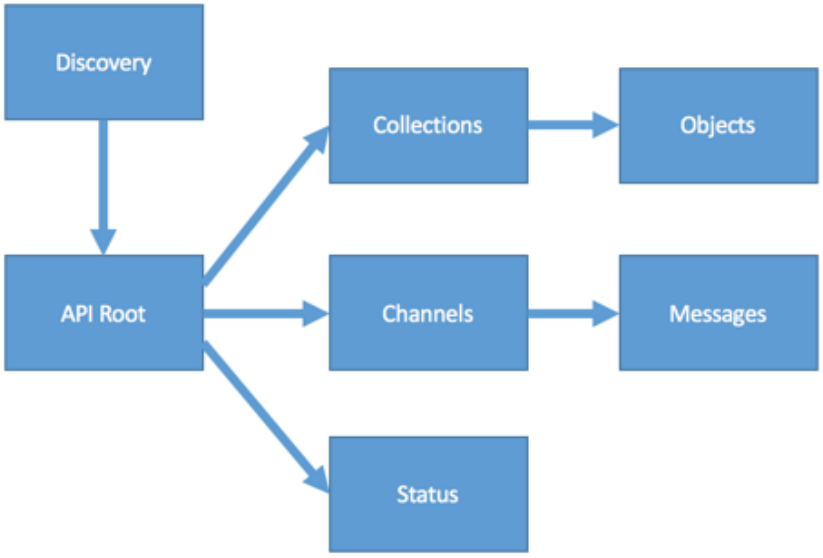
Figure 1.1 summarizes the relationships between the components of an API Root.

**Example API Root URLs**

```
https://example.com/
https://api1.example.com/
https://example.com/api1/
https://example.com/api2/
https://example.org/trustgroup1/
https://example.org/taxii2/api1/
```

Figure 1.1




### 1.6.3 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.

### 1.6.4 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).

### 1.6.5 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

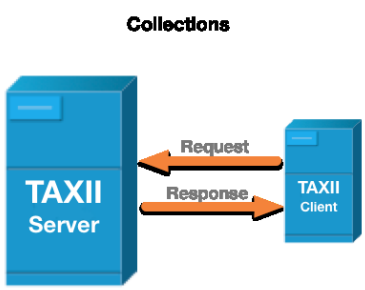
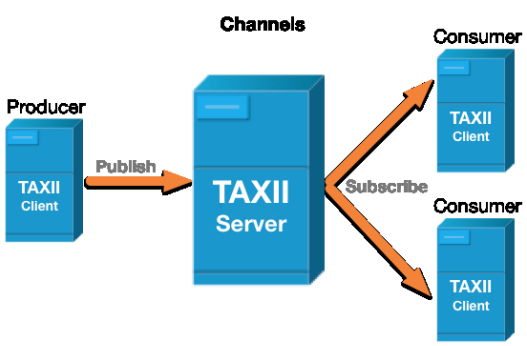



Figure 1.3




### 1.6.6 Transport

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

### 1.6.7 Serialization

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

### 1.6.8 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].

#### 1.6.8.1 Media Types

The STIX and TAXII media types used in this specification are summarized in the following table and are used for both requests and responses. Appendix B contains the official IANA registration information for the TAXII media type

| Type | Subtype | Example |
|---|---|---|
| application | taxii+json | application/taxii+json |
| application | stix+json | application/stix+json |

### 1.6.8.2 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] and the value is of the form 'n.m', where n is the major version and m the minor version, both unsigned integer values.

The value for the version parameter that represents the final contents 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 | TAXII version 2.1 in JSON |
| application/taxii+json | Latest version of TAXII that the server supports |
| application/stix+json;version=2.1 | STIX version 2.1 in JSON |
| application/stix+json | Latest version of STIX that the server supports |

## 1.6.9 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 suggested authentication scheme in TAXII. As specified in sections 8.2.2 and 8.5.1, TAXII Servers are encouraged to implement support for HTTP Basic 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, then the TAXII Server will respond with an 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 properties from any response if the authenticated client is not authorized to receive them, so long as that omission does not violate this specification.

### 1.6.10 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.

### 1.6.11 Object Lifecycle

There are no requirements defined in this specification for how long a TAXII server needs to store a specific object on the server or within a collection after it has been successfully posted to a TAXII collection. If the TAXII server chooses to remove an entire object or any number of versions of the object from the server or collection that is entirely up to the software, its deployment, and the use cases it supports.

## 1.7 Changes From Earlier Versions

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

### 1.7.1 TAXII 2.1 Major Changes and Additions

TAXII 2.1 differs from TAXII 2.0 in the following ways:

1. The DNS SRV record was changed from taxii to taxii2
2. The discovery URL was changed from /taxii/ to /taxii2/
3. The Manifest Resource was changed to represent individual versions of an object, instead of an object with all of its versions
4. Item based pagination was removed from this version of the specification

5. The section on content negotiation was updated
6. The media types were changed throughout the document
7. Clarification was added to say that API Roots can be relative paths as well as absolute paths
8. Changed version value in API Roots to match media type
9. Changed status resource to allow status on success and pending
10. Add TAXII media type as Accept type in 5.4 and 5.6 since a TAXII error message could be returned
11. HTTP Basic is now a SHOULD implement for the Server
12. Added a DELETE object by ID endpoint
13. Added a versions endpoint for object by ID.
14. Added section on Server Implementation Considerations

Please see Appendix C for a list GitHub issues that were resolved for this release.

# 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 properties 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. |

| | |
|---|---|
| | 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.<br><br>The JSON MTI serialization uses the JSON array type [RFC8259], which is an ordered list of values. |
| `manifest` | A Manifest Resource, see section 5.3.1. |
| `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` property **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. |
| `versions` | A Versions Resource, see section 5.8.1. |

# 3 TAXII™ - 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
- 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) **MAY** contain an `error` message (see section 3.5.1) in the response body.
- All TAXII requests **MUST** include a media range in the `Accept` header and **MUST** include at least one TAXII media range. Requests for TAXII or STIX content **MUST** use the values from section 1.6.8 and **SHOULD** include the optional version parameter defined in that section.
- 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.
- Requests with media types in the `Accept` headers that are defined for that Endpoint **MUST NOT** result in an HTTP 406 (Not Acceptable)response.
- Requests with a media type in the `Content-Type` header that is defined for that Endpoint **MUST NOT** result in an HTTP 415 (Unacceptable Media Type) response.
- Requests with media types in the `Accept` headers that are not defined for that Endpoint **MAY** be satisfied with the appropriate content or **MAY** result in an HTTP 406 (Not Acceptable) response.
- Requests with a media type in the `Content-Type` header that is not defined for that Endpoint **MAY** be satisfied with the appropriate content or **MAY** result in an HTTP 415 (Unacceptable Media Type) response.
- All TAXII POST requests **MUST** include a valid `Accept` and `Content-Type` header.
- 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 |
|---|---|---|
| **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}**/manifest/ | GET | `manifest` |
| **{api-root}**/collections/**{id}**/objects/ | GET, POST | `object`* |
| **{api-root}**/collections/**{id}**/objects/**{object-id}**/ | GET, DELETE | `object`* |
| **{api-root}**/collections/**{id}**/objects/**{object-id}**/versions/ | GET | `versions` |
| **Channels (section 6)** | | |
| TBD in a future version | | |

** The actual format of objects is dependent on HTTP Content negotiation, as discussed in section 1.6.8*

## 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 series 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>• GET **{api-root}**/collections/**{id}**/objects/**{object-id}**/versions/<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 series 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>• GET **{api-root}**/collections/**{id}**/objects/**{object-id}**/versions/<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. |

| | |
|---|---|
| | 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`<br>`?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. |

| | |
|---|---|
| | 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].<br>  ○ 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 properties 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 property 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`) 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 `taxii2`. As defined in [RFC2782], the service name is defined without an underscore, and an underscore is added to construct the correct name in the actual DNS SRV record. The protocol for this DNS SRV record **MUST** be tcp.

**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:

```
_taxii2._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 1.6.1 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/**

### Implementation Notes

Get information about the TAXII Server and any advertised API Roots

### Requests

#### Required Headers

```
Accept: application/taxii+json;version=2.1
```

### Successful Responses

#### Response Codes

**200** - The request was successful

### Required Headers

```
Content-Type: application/taxii+json;version=2.1
```

### Payload

`discovery`

## Failure Responses

### Response Codes

**401** - The client needs to authenticate

**403** - The client does not have access to this resource

**404** - The Discovery service is not found or the client does not have access to the resource

**406** - The media type provided in the Accept header is invalid

### Required Headers

```
Content-Type: application/taxii+json;version=2.1
```

### Payload

`error`

## Example

### 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/"
      ]
    }
```

### 4.1.1 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 property 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 |

| | | |
|---|---|---|
| | | //someserver.example.com/apiroot1<br>../someapiroot/<br>https://foo.edu/bar?baz |

## 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}/`

### Implementation Notes

Get information about a specific API Root

### Requests

#### URL Parameters

`{api-root}` - the base URL of the API Root

#### Required Headers

```
Accept: application/taxii+json;version=2.1
```

### Successful Responses

#### Response Codes

**200** - The request was successful

#### Required Headers

```
Content-Type: application/taxii+json;version=2.1
```

#### Payload

`api-root`

## Failure Responses

### Response Codes

**401** - The client needs to authenticate

**403** - The client does not have access to this resource

**404** - The API Root is not found or the client does not have access to the resource

**406** - The media type provided in the Accept header is invalid

### Required Headers

```
Content-Type: application/taxii+json;version=2.1
```

### Payload

`error`

## Example

### 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": ["application/taxii+json;version=2.1"],
  "max_content_length": 104857600
}
```

### 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 1.6.8.1 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. |

## 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 5.5). 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}/`

### Implementation Notes

Get status information for a specific status ID

### Requests

**URL Parameters**

`{api-root}` - the base URL of the API Root

`{status-id}` - the identifier of the status message being requested

#### Required Headers

```
Accept: application/taxii+json;version=2.1
```

## Successful Responses

#### Response Codes

**200** - The request was successful

#### Required Headers

```
Content-Type: application/taxii+json;version=2.1
```

#### Payload

`status`

## Failure Responses

#### Response Codes

**401** - The client needs to authenticate

**403** - The client does not have access to this resource

**404** - The API Root or Status ID are not found or the client does not have access to the resource

**406** - The media type provided in the Accept header is invalid

#### Required Headers

```
Content-Type: application/taxii+json;version=2.1
```

#### Payload

`error`

## Example

#### 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": [
    {
      "id": "indicator--c410e480-e42b-47d1-9476-85307c12bcbf" ,
      "version": "2018-05-27T12:02:41.312Z"
    }
  ],
  "failure_count": 1,
  "failures": [
    {
      "id": "malware--664fa29d-bf65-4f28-a667-bdb76f29ec98",
      "version": "2018-05-28T14:03:42.543Z",
      "message": "Unable to process object"
    }
  ],
  "pending_count": 2,
  "pendings": [
    {
      "id": "indicator--252c7c11-daf2-42bd-843b-be65edca9f61",
      "version": "2018-05-18T20:16:21.148Z"
    },
    {
      "id": "relationship--045585ad-a22f-4333-af33-bfd503a683b5",
      "version": "2018-05-15T10:13:32.579Z"
    }
  ]
}
```

### 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 POST request (see section 5.5) 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 failed to be added, are still pending, or have been successfully added is a simple type that contains the identifier of the object (e.g., for STIX objects, their **`id`**), its version, and an optional message indicating additional details.

| 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 `status-details` | A list of objects that were successfully processed. |
| **`failure_count`** (required) | `integer` | The number of objects that failed to be created. |
| **`failures`** (optional) | `list` of type `status-details` | 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 status-details | A list of objects that have yet to be processed. |

**Type Name:** status-details

This type represents an object that was added, is pending, or not added to the Collection. It contains the **id** and **version** of the object along with a **message** describing any details about its status.

| Property Name | Type | Description |
|---|---|---|
| **id** (required) | string | The identifier of the object that succeed, is pending, or 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**. |
| **version** (required) | string | The version of the object that succeeded, is pending, or failed to be created. For STIX objects the version **MUST** be the STIX modified timestamp Property. If a STIX object is not versioned (and therefore does not have a modified timestamp) the created timestamp **MUST** be used. |
| **message** (optional) | string | A message indicating more information about the object being created, its pending state, or why the object failed to be created. |

**Status URL Examples**

```
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/
```

# 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}**/manifest/ | GET | `manifest` |
| **{api-root}**/collections/**{id}**/objects/ | GET, POST | `object` |
| **{api-root}**/collections/**{id}**/objects/**{object-id}**/ | GET, DELETE | `object` |
| **{api-root}**/collections/**{id}**/objects/**{object-id}**/versions/ | GET | `versions` |

## 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/`

### Implementation Notes

Get information about all collections

## Requests

### URL Parameters

`{api-root}` - the base URL of the API Root

### Required Headers

`Accept: application/taxii+json;version=2.1`

## Successful Responses

### Response Codes

**200** - The request was successful

### Required Headers

`Content-Type: application/taxii+json;version=2.1`

### Payload

`collections`

## Failure Responses

### Response Codes

**401** - The client needs to authenticate

**403** - The client does not have access to this resource

**404** - The API Root is not found or the client does not have access to the resource

**406** - The media type provided in the Accept header is invalid

### Required Headers

`Content-Type: application/taxii+json;version=2.1`

### Payload

`error`

## Example

### 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 contains 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"
      ]
    }
  ]
}
```

### 5.1.1 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. |

## 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}/`

### Implementation Notes

Get information about a specific collection

### Requests

#### URL Parameters

`{api-root}` - the base URL of the API Root

`{id}` - the `identifier` of the Collection being requested

#### Required Headers

`Accept: application/taxii+json;version=2.1`

### Successful Responses

#### Response Codes

**200** - The request was successful

#### Required Headers

`Content-Type: application/taxii+json;version=2.1`

#### Payload

`collection`

### Failure Responses

#### Response Codes

**401** - The client needs to authenticate

**403** - The client does not have access to this resource

**404** - The API Root or Collection ID are not found or the client does not have access to the resource

**406** - The media type provided in the Accept header is invalid

#### Required Headers

```
Content-Type: application/taxii+json;version=2.1
```

#### Payload

`error`

### Example

#### 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 contains high value IOCs",
  "can_read": true,
  "can_write": false,
  "media_types": [
    "application/stix+json;version=2.1"
  ]
}
```

## 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 property is equivalent to a single-value list containing "application/stix+json". This list **MUST** describe all media types that the Collection can store. |

## 5.3 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 5.4) 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** property.

## GET /{api-root}/collections/{id}/manifest/

### Implementation Notes

Get manifest information about the contents of a specific collection.

### Requests

#### URL Parameters

{api-root} - the base URL of the API Root
{id} - the identifier of the Collection being requested

#### URL Filtering Parameters

added_after - a timestamp (e.g., ?added_after=...)
id - an id of an object (e.g., ?match[id]=...)
type - the type of an object (e.g., ?match[type]=...)
version - the version of an object (e.g., ?match[version]=...)

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.

#### Required Headers

Accept: application/taxii+json;version=2.1,application/stix+json;version=2.1

### Successful Responses

#### Response Codes

**200** - The request was successful

#### Required Headers

Content-Type: application/taxii+json;version=2.1
X-TAXII-Date-Added-First: timestamp
X-TAXII-Date-Added-Last: timestamp

#### Payload

manifest

### Failure Responses

### Response Codes

**401** - The client needs to authenticate

**403** - The client does not have access to this resource

**404** - The API Root or Collection ID are not found or the client does not have access to the resource

**406** - The media type provided in the Accept header is invalid

### Required Headers

Content-Type: application/taxii+json;version=2.1

### Payload

`error`

## Example

### 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
X-TAXII-Date-Added-First: 2018-05-12T03:05:09.15811Z
X-TAXII-Date-Added-Last: 2018-05-12T03:05:17.275544Z

{
  "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"
          }
        ]
      }
```

### 5.3.1 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) | `string` | The identifier of the object that this manifest entry describes. 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`**. |
| **`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`** property 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. |

## 5.4 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 `objects` 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/`

### Implementation Notes

Get all objects from a collection

### Requests

#### URL Parameters

`{api-root}` - the base URL of the API Root
`{id}` - the `identifier` of the Collection being requested

#### URL Filtering Parameters

```
added_after - a timestamp              (e.g., ?added_after=...)
id          - an id of an object       (e.g., ?match[id]=...)
type        - the type of an object    (e.g., ?match[type]=...)
version     - the version of an object (e.g., ?match[version]=...)
```

#### Required Headers

```
Accept: application/taxii+json;version=2.1,application/stix+json;version=2.1
```

### Successful Responses

#### Response Codes

**200** - The request was successful

#### Required Headers

Content-Type: application/stix+json;version=2.1

X-TAXII-Date-Added-First: timestamp

X-TAXII-Date-Added-Last: timestamp

#### Payload

bundle

## Failure Responses

#### Response Codes

**401** - The client needs to authenticate

**403** - The client does not have access to this resource

**404** - The API Root or Collection ID are not found or the client does not have access to the resource

**406** - The media type provided in the Accept header is invalid

#### Required Headers

Content-Type: application/taxii+json;version=2.1

#### Payload

error

## Example

#### GET Request

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

#### GET Response

```
HTTP/1.1 200 OK
Content-Type: application/stix+json;version=2.1
X-TAXII-Date-Added-First: 2018-05-12T03:05:09.15811Z
X-TAXII-Date-Added-Last: 2018-05-12T03:05:17.275544Z
```

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

## 5.5 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 a client publishes an exact duplicate of an Object already present in the Collection, the server **MUST** not return an error for that Object.

If the request is marked `pending` in the **`status`** property, 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/**

### Implementation Notes

Add a new object to a specific collection

### Requests

### URL Parameters

{api-root} - the base URL of the API Root

{id} - the identifier of the Collection where objects are being added

### Required Headers

Accept: application/taxii+json;version=2.1

Content-Type: application/stix+json;version=2.1

### Payload

bundle

## Successful Responses

### Response Codes

**202** - The request was successful accepted

### Required Headers

Content-Type: application/taxii+json;version=2.1

### Payload

status

## Failure Responses

### Response Codes

**401** - The client needs to authenticate

**403** - The client does not have access to write to this resource

**404** - The API Root or Collection ID are not found or the client can not write to this resource

**406** - The media type provided in the Accept header is invalid

**413** - The POSTed payload exceeds the max_content_length of the API Root

**415** - The client attempted to POST a payload with a content type the server does not support

**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

### Required Headers

Content-Type: application/taxii+json;version=2.1

### Payload

`error`

## Example

### 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": [
    {
      "id": "indicator--c410e480-e42b-47d1-9476-85307c12bcbf"
    }
  ],
```

```
        "failure_count": 0,
        "pending_count": 3
    }
```

## 5.6 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}/**

### Implementation Notes

Get a specific object from a collection

### Requests

#### URL Parameters

`{api-root}` - the base URL of the API Root
`{id}` - the `identifier` of the Collection being requested
`{object-id}` - the ID of the object being requested

#### URL Filtering Parameters

added_after - a `timestamp` (e.g., ?added_after=...)
version - the version of an object (e.g., ?match[version]=...)

#### Required Headers

```
Accept: application/taxii+json;version=2.1,application/stix+json;version=2.1
```

## Successful Responses

### Response Codes

**200** - The request was successful

### Required Headers

Content-Type: application/stix+json;version=2.1

X-TAXII-Date-Added-First: timestamp

X-TAXII-Date-Added-Last: timestamp

### Payload

bundle

## Failure Responses

### Response Codes

**401** - The client needs to authenticate

**403** - The client does not have access to this resource

**404** - The API Root, Collection ID and/or Object ID are not found or the client does not have access to the resource

**406** - The media type provided in the Accept header is invalid

### Required Headers

Content-Type: application/taxii+json;version=2.1

### Payload

error

## Example

### 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,application/taxii+json;version=2.1
```

### GET Response

```
HTTP/1.1 200 OK
```

```
Content-Type: application/stix+json;version=2.1
X-TAXII-Date-Added-First: 2018-05-12T03:05:09.15811Z
X-TAXII-Date-Added-Last: 2018-05-12T03:05:17.275544Z

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

## 5.7 Delete an Object

This Endpoint deletes an object from a Collection by its **id**. For STIX 2 objects, the `{object-id}` **MUST** be the STIX **id**.

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.

If the Collection specifies `can_write` as `true` and `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. If a DELETE operation were to work on a write only collection, the result would leak information about the presence, or lack thereof, of an object in the collection.

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

When a TAXII Server returns a successful response code of 200 to a DELETE, any subsequent attempts to fetch that object from the collection using the Get Objects Endpoint **MUST** return either an HTTP 401 (Unauthorized) or an HTTP 404 (Not Found) response. Furthermore, any subsequent attempts to fetch the collection manifest or its contents **MUST NOT** return the object in the result. Subsequent attempts to fetch the object from other collections present on the TAXII server **MAY** also return an HTTP 404 (Not Found) response, and subsequent attempts to fetch the collection manifest or its contents of other collections which contained the object **MAY** not return the object in the result.

**DELETE** **/{api-root}/collections/{id}/objects/{object-id}/**

## Implementation Notes

Delete a specific object from a collection

## Requests

### URL Parameters

`{api-root}` - the base URL of the API Root

`{id}` - the `identifier` of the Collection being requested

`{object-id}` - the ID of the object being requested

### URL Filtering Parameters

version - the version of an object (e.g., ?match[version]=...)

### Required Headers

Accept: application/taxii+json;version=2.1

## Successful Responses

### Response Codes

**200** - The request was successful

### Required Headers

Content-Type: application/stix+json;version=2.1

### Payload

n/a

## Failure Responses

### Response Codes

**401** - The client needs to authenticate

**403** - The client does not have access to this resource

**404** - The API Root, Collection ID and/or Object ID are not found or the client does not have access to the resource

**406** - The media type provided in the Accept header is invalid

### Required Headers

Content-Type: application/taxii+json;version=2.1

#### Payload

`error`

## Example

#### DELETE Request

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

#### DELETE Response

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

## 5.8 Get Object Versions

This Endpoint retrieves a list of versions of an object in a Collection. This list can be used to decide whether it's worth retrieving the actual objects, or if new versions have been added.

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, like the last or first versions of an object, this Endpoint supports filtering as defined in section 3.4. The only valid match parameter is `version`.

**GET** `/{api-root}/collections/{id}/objects/{object-id}/versions/`

## Implementation Notes

Get a list of object versions from a collection

## Requests

#### URL Parameters

`{api-root}` - the base URL of the API Root

`{id}` - the `identifier` of the Collection being requested

`{object-id}` - the ID of the object being requested

### URL Filtering Parameters

added_after - a timestamp (e.g., ?added_after=...)

version - the version of an object (e.g., ?match[version]=...)

### Required Headers

Accept: application/taxii+json;version=2.1

## Successful Responses

### Response Codes

**200** - The request was successful

### Required Headers

Content-Type: application/taxii+json;version=2.1

X-TAXII-Date-Added-First: timestamp

X-TAXII-Date-Added-Last: timestamp

### Payload

versions

## Failure Responses

### Response Codes

**401** - The client needs to authenticate

**403** - The client does not have access to this resource

**404** - The API Root, Collection ID and/or Object ID are not found or the client does not have access to the resource

**406** - The media type provided in the Accept header is invalid

### Required Headers

Content-Type: application/taxii+json;version=2.1

### Payload

error

## Example

**GET Request**

```
GET /api1/collections/91a7b528-80eb-42ed-a74d-c6fbd5a26116/object/
indicator--252c7c11-daf2-42bd-843b-be65edca9f61/versions/ 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
X-TAXII-Date-Added-First: 2018-05-12T03:05:09.15811Z
X-TAXII-Date-Added-Last: 2018-05-12T03:05:17.275544Z

{
  "versions": [
    "2016-11-03T12:30:59.000Z",
    "2016-11-03T12:31:00.000Z"
  ]
}
```

### 5.8.1 Versions Resource

**Resource Name:** `versions`

The `versions` resource is a simple wrapper around a list of versions.

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

# 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 **MUST** only contain the characters a–z (lowercase ASCII), 0–9, and underscore (_).
- Custom Property names **SHOULD** start with "x_" followed by a source unique identifier (such as a domain name with dots replaced by underscores), an underscore and then the name. For example, **`x_example_com_customfield`**.
- Custom Property names **MUST** have a minimum length of 3 ASCII characters.
- Custom Property names **MUST** be no longer than 250 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 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 **SHOULD** 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 1.6.9.
8. It **MAY** restrict access to clients by omitting specific objects, information, or optional properties 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 1.6.9.

### 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. IANA Considerations

This appendix contains the required information to register the TAXII media type with IANA. While some of the information here is only for IANA, implementers of TAXII should pay close attention to the security considerations and privacy considerations outlined in this appendix.

This document defines the "application/taxii+json" media type

Media type name: application

Media subtype name: taxii+json

Required parameters: None

Optional parameters: version

> This parameter is used to designate the specification version of TAXII that is being used during HTTP content negotiation. Example: "application/taxii+json;version=2.1". The parameter value is of the form 'n.m', where n is the major version and m the minor version, both unsigned integer values.

Encoding considerations: binary

> Encoding considerations are identical to those specified for the "application/json" media type. See [RFC8259].

Security considerations:

> These considerations are, in part, derived from Section 9 of the Resource-Oriented Lightweight Information Exchange [RFC8322].
>
> This document defines a resource-oriented approach for exchanging cyber threat intelligence using HTTP [RFC7230] over TLS [RFC5246] and the JSON [RFC8259] Format. As such, implementers must understand the security considerations described in those specifications.
>
> Security considerations relating to the generation and consumption of TAXII messages are similar to application/json and are discussed in Section 12 of [RFC8259].
>
> The discovery API contains one or multiple URLs, therefor the security considerations stated in Section 6 of [RFC1738] should be consulted especially in regards to parsing relative URLs and attempts of path traversal.
>
> Documents of "application/taxii+json" are simply request and response messages for an RPC like mechanism for searching, uploading and downloading Cyber Threat Intelligence (CTI) documents, most commonly STIX. The documents only contain metadata about the TAXII server, such as descriptions, versions of CTI or status response of the request. Documents do not contain active or executable content.

Unicode is used to represent text such as descriptions in the format. The considerations documented by Unicode Technical Report #36: Unicode Security Considerations [UNICODE] should be taken into account.

To protect the confidentiality of a given resource provided by a TAXII implementation, requests for retrieval of a resource need to be authenticated to prevent unauthorized users from accessing the resource. It can also be useful to log and audit access to sensitive resources to verify that proper access controls remain in place over time.

Access control to content made available using TAXII should use mechanisms that are appropriate to the sensitivity of the information. While the primitive authentication mechanism of HTTP Basic Authentication [RFC7617] is mandatory to implement for base level interoperability it is rarely appropriate for sensitive information. A number of authentication schemes are defined in the "HTTP Authentication Schemes" registry at IANA [IANA AUTH]. Of these, HTTP Origin-Bound Authentication (HOBA) [RFC7486] and SCRAM-SHA-256 [RFC7804] ("SCRAM" stands for "Salted Challenge Response Authentication Mechanism") provide improved security properties over HTTP Basic [RFC7617] and Digest [RFC7616] authentication schemes. However, sharing communities that are engaged in sensitive collaborative analysis and/or operational response for indicators and incidents targeting high-value information systems should adopt a suitably stronger user authentication solution, such as a risk-based or multi-factor approach.

Collaborating consortiums may benefit from the adoption of a federated identity solution, such as those based upon OAuth [RFC6749] with the JSON Web Token (JWT) [RFC7797], or SAML-core [SAML-core] ("SAML" stands for "Security Assertion Markup Language"), SAML-bind [SAML-bind], and SAML-prof [SAML-prof] for web-based authentication and cross-organizational single sign-on. Dependency on a trusted third-party identity provider implies that appropriate care must be exercised to sufficiently secure the identity provider. Any attacks on the federated identity system would present a risk to the consortium, as a relying party. Potential mitigations include deployment of a federation-aware identity provider that is under the control of the information-sharing consortium, with suitably stringent technical and management controls.

It is recommended that all TAXII servers authenticate and authorize access to all collection data on a per-client basis using robust security methods. While this specification defines HTTP Basic as a minimum suggested authentication mechanism, more advanced security authentication methods are recommended when products or deployments require stronger authentication and authorization frameworks for accessing or posting data to the TAXII server.

Authorization of resource representations is the responsibility of the source system, i.e., based on the authenticated user identity associated with an HTTP(S) request. The required authorization policies that are to be enforced must therefore be managed by the security administrators of the source system. Various authorization architectures would be suitable for this purpose, such as Role-Based Access Control (RBAC) [NIST RBAC] and/or Attribute-Based Access Control (ABAC), as embodied in the eXtensible Access Control Markup Language (XACML) [XACML]. In particular, implementers adopting XACML may benefit from the capability to represent their authorization policies in a standardized, interoperable format. Note that implementers are free to choose any suitable authorization mechanism that is capable of fulfilling the policy enforcement requirements relevant to their consortium and/or organization.

While the authentication and confidentiality for the TAXII session is done at a lower level via the transport mechanism (HTTPS), this does not obviate the consumer (server or client) from validating the format and contents of the documents sent in a session. This validation should

includes checking various limits, such as document size limits, to limit the risk of the other party attempting to attack the service.

Additional security requirements such as enforcing message-level security at the destination system could supplement the security enforcements performed at the source system; however, these destination-provided policy enforcements are out of scope for this specification. Implementers requiring this capability should consider leveraging, for example, the <RIDPolicy> element in the RID schema. Refer to Section 9 of [RFC6545] for more information. Additionally, the underlying JSON serialization used in the representation can offer encryption and message authentication capabilities. For example, JSON Web Encryption [RFC7516] and JSON Web Signature [RFC7515], can provide such mechanisms.

TAXII Servers may manage response volume in different ways. Implementers should be aware that a search request may return more records than is prudent to return in a single HTTP Response. To mitigate this, TAXII servers should consider implementing restrictions on the number of records it will return in a single HTTP response.

TAXII provides clients a rich set of filtering and query options to return specific results from repositories of CTI data. As such, TAXII servers should implement protections against queries that can potentially consume a significant amount of resources and prevent the server from functioning in a normal way.

TAXII defines an optional error message that may contain sensitive application data. Implementers should ensure that they do not leak descriptive text or application return codes for things that a user may not have access to, for example leak info about existence of something, implementation of something, vs. just not having access.

Note: Despite TAXII searching and returning STIX objects, this format does not encapsulate any CTI content. It is expected that CTI documents will be sent with the appropriate mime-type. For these, consult their own security consideration sections.

Privacy considerations

These considerations are, in part, derived from Section 10 of the Resource-Oriented Lightweight Information Exchange [RFC8322].

The documents contain various descriptions and other text. There is no expectation that these will contain private information, but as some may be user provided, there is no guarantee that a user will not inadvertently include private data. It is expected that the client or server authenticate the other party through the transport mechanism before sending any possible private data. As the protocol is about sharing data, it is expected that the parties understand their obligations in keeping relevant data private.

Adoption of the information-sharing approach described in this document will enable users to more easily perform correlations across separate, and potentially unrelated, cybersecurity information providers. A client may succeed in assembling a data set that would not have been permitted within the context of the authorization policies of either provider when considered individually. Thus, providers may face a risk of an attacker obtaining an access that constitutes an undetected separation of duties (SOD) violation. It is important to note that this risk is not unique to this specification, and a similar potential for abuse exists with any other cybersecurity information-sharing protocol. However, the wide availability of tools for HTTP clients implies that the resources and technical skills required for a successful exploit may be less than it was

previously. This risk can be best mitigated through appropriate vetting of the client at the time of account provisioning. In addition, any increase in the risk of this type of abuse should be offset by the corresponding increase in effectiveness that this specification affords to the defenders.

Overall, privacy concerns in TAXII can be mitigated by following security considerations and by the careful use of the content exchanged via TAXII.

Interoperability considerations:

The TAXII specification specifies the format of conforming messages and the interpretation thereof. In addition, the OASIS Cyber Threat Intelligence (CTI) Technical Committee has defined interoperability tests to ensure conforming products and solutions can exchange TAXII documents.

Published specification:

TAXII Version 2.0 OASIS Committee Specification 01

http://docs.oasis-open.org/cti/taxii/v2.0/cs01/taxii-v2.0-cs01.html

Cited in the "OASIS Standards" document:

https://www.oasis-open.org/standards#oasiscommiteespecs, from

https://www.oasis-open.org/standards#taxii2.0

Applications which use this media:

TAXII is an application layer protocol for the communication of cyber threat information including STIX in a simple and scalable manner.

Fragment identifier considerations: None

Restrictions on usage: None

Additional information:

1. Deprecated alias names for this type: application/vnd.oasis.taxii+json
2. Magic number(s): n/a [RFC8259]
3. File extension(s): None
4. Macintosh file type code: TEXT [RFC8259]
5. Object Identifiers: None

Person and email to contact for further information: Chet Ensign (chet.ensign@oasis-open.org)

Intended usage: COMMON

Author:

OASIS Cyber Threat Intelligence (CTI) Technical Committee;

URI reference: http://www.oasis-open.org/committees/cti/.

Change controller: OASIS

Provisional registration: No

# Appendix C. 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)

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

Richard Struse, 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:

Qian Yin, 360 Enterprise Security Group

Xinhua Zheng, 360 Enterprise Security Group

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
RaphaÎl Vinot, CIRCL
Syam Appala, Cisco Systems
Ted Bedwell, Cisco Systems
David McGrew, 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 SÓrghi, 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.
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
Iain Brown, GDS
Adam Cooper, GDS
James Penman, GDS
Howard Staple, GDS
Chris Taylor, GDS
Laurie Thomson, GDS
Alastair Treharne, GDS
Julian White, 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
Nick Rossmann, IBM
Laura Rusu, IBM
Ron Williams, IBM
Paul Martini, iboss, Inc.
Vasileios Mavroeidis, IFI
Joerg Eschweiler, Individual
Stefan Hagen, Individual
Elysa Jones, Individual
Terry MacDonald, 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
Sarah Kelley, 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
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
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
Cedric LeRoux, Splunk Inc.
Brian Luger, Splunk Inc.
Philip Royer, Splunk Inc.
Sourabh Satish, Splunk Inc.
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.
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.
Kris Anderson, Trend Micro
David Girard, Trend Micro

Eric Shulze, Trend Micro
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
James Bohling, 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
Lee Chieffalo, Viasat
Wilson Figueroa, Viasat
Andrew May, Viasat
Ales Cernivec, XLAB
Anthony Rutkowski, Yanna Technologies LLC

# Appendix D. Revision History

| Revision | Date | Editor(s) | Changes Made |
|---|---|---|---|
| 01 | 2018-04-10 | Bret Jordan<br>Drew Varner | Initial Version<br><br>GitHub Issues: 1, 8, 9, 10, 11, 12, 17, 18, 19, 20, 21, 23, 25, 26, 28, 29, 30, 31, 34, 35, 36, 39, 41, 43, 48, 49, 51, 53, 54, 56, 57, 62, 63 |
| 02 | 2018-05-18 | Bret Jordan<br>Drew Varner | Fix typos, add clarifying text in some cases, changed DNS SRV record name, changed status resource, and made other minor changes.<br><br>GitHub Issues: 47, 65, 66, 67, 70, 73, 74, 75 |
| 03 | 2018-07-12 | Bret Jordan<br>Drew Varner | Reformatted the endpoint definitions, Added an endpoint to DELETE an object, and added a versions endpoint to object by ID. Added IANA Considerations Appendix. Changed angle brackets "<>" in the URL parameters to be curly brackets "{}" to better align with API documentation best practices. Added non-normative references section.<br><br>GitHub Issues: 37, 38, 58, 76, 78, 79, 80, 82, 84 |