IIIF Presentation API 3.0 ALPHA DRAFT

Status of this Document

This Version: 3.0.0-ALPHA

Latest Stable Version: 2.1.1 (/api/presentation/2.1/)

Previous Version: 2.1.1 (/api/presentation/2.1/)

Editors

Michael Appleby (https://orcid.org/0000-0002-1266-298X) (https://orcid.org/0000-0002-1266-298X), *Yale University* (http://www.yale.edu/)
Tom Crane (https://orcid.org/0000-0003-1881-243X) (https://orcid.org/0000-0003-1881-243X), *Digirati* (http://digirati.com/)
Robert Sanderson (https://orcid.org/0000-0003-4441-6852) (https://orcid.org/0000-0003-4441-6852), *J. Paul Getty Trust* (http://www.getty.edu/)
Jon Stroop (https://orcid.org/0000-0002-0367-1243) (https://orcid.org/0000-0002-0367-1243), *Princeton University Library* (https://library.princeton.edu/)
Simeon Warner (https://orcid.org/0000-0002-7970-7855) (https://orcid.org/0000-0002-7970-7855), *Cornell University* (https://www.cornell.edu/)

Status Warning This is a work in progress and may change without any notices. Implementers should be aware that this document is not stable. Implementers are likely to find the specification changing in incompatible ways. Those interested in implementing this document before it reaches beta or release stages should join the mailing list (mailto:iiif-discuss@googlegroups.com) and take part in the discussions, and follow the emerging issues (https://github.com/IIIF/iiif.io/milestone/8) on Github.

Table of Contents

- 1. Introduction
  - 1.1. Objectives and Scope
  - 1.2. Motivating Use Cases
  - 1.3. Terminology
- 2. Resource Type Overview
  - 2.1. Defined Types
  - 2.2. Additional Types
- 3. Resource Properties
  - 3.1. Descriptive Properties
  - 3.2. Technical Properties
  - 3.3. Linking Properties
    - 3.3.1. External Links

3.4.2. Internal Links
3.4. Structural Properties
3.6. Values
4. JSON-LD (JSON for Linking Data) Considerations
4.1. Resource Representations
4.2. Properties with Multiple Values
4.3. Language of Property Values
4.4. HTML (HyperText Markup Language) Markup in Property Values
4.5. Linked Data Context and Extensions
5. Resource Structure
5.1. Collection
5.2. Manifest
5.3. Canvas
5.4. Range
5.5. Annotation Pages
5.6. Annotations
5.7. Annotation Collection
6. HTTP (Hypertext Transfer Protocol) Requests and Responses
6.1. URI Recommendations
6.2. Requests
6.3. Responses
7. Authentication
Appendices
A. Summary of Metadata Requirements
B. Example Manifest Response
C. Versioning
D. Acknowledgements
E. Change Log

1. Introduction

Access to digital representations of structured resources is a fundamental requirement for many research activities, the transmission of cultural knowledge, and for the daily pursuits of every web citizen. Digital content is the primary mode of transmission for access to cultural heritage, science and entertainment. Collections of both digitized physical objects and much born-digital content benefit from a standardized description of their structure, layout and presentation mode.

This document describes how the structure and layout of composite objects can be made available in a standard manner. Many different styles of viewer can be implemented that consume the information to enable a rich and dynamic experience, presenting content from across collections and institutions.

A composite object may comprise a series of pages, surfaces or other views; for example the single view of a painting, the two sides of a photograph, four cardinal views of a statue, or the many pages of an edition of a newspaper or book. The primary requirements for this specification are to provide an order for these views, the resources needed to display a representation of the view, and the descriptive information needed to allow the user to understand what is being seen.

The principles of Linked Data (http://linkeddata.org/) and the Architecture of the Web (http://www.w3.org/TR/webarch/) are adopted in order to provide a distributed and interoperable system. The Shared Canvas data model (/model/shared-canvas/1.0) and JSON-LD (JSON for Linking Data) (http://www.w3.org/TR/json-ld/) are leveraged to create an easy-to-implement, JSON (JavaScript Object Notation)-based format.

Please send feedback to iiif-discuss@googlegroups.com (mailto:iiif-discuss@googlegroups.com)

1.1. Objectives and Scope

The objective of the IIIF (International Image Interoperability Framework) (pronounced “Triple-Eye-Eff”) Presentation API (Application Programming Interface) is to provide the information necessary to allow a rich, online viewing environment for structured digital objects to be presented to a human user, often in conjunction with the IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) (/api/image/2.1/). This is the sole purpose of the API (Application Programming Interface) and therefore the descriptive information is given in a way that is intended for humans to read, but not semantically available to machines. In particular, it explicitly does **not** aim to provide metadata that would drive a search engine for discovering unknown objects.

Implementations of this specification will be able to:

- display to the user digitized images, video, audio and other content types associated with a particular physical or born-digital object ;

- allow the user to navigate between multiple views of the object, either sequentially or hierarchically ;
- display descriptive information about the object, view or navigation structure to provide context to the user ;
- and provide a shared environment in which both publishers and users can annotate the object and its content with additional information.

The following are **not** in scope:

- The discovery or selection of interesting objects is not directly supported; however hooks to reference further resources are available.
- Search within the object; which is described by the IIIF (International Image Interoperability Framework) Content Search API (Application Programming Interface) (/api/search/1.0/).

Note that in the following descriptions, “object” is used to refer to the object that has been digitized or a born-digital compound object, and “resources” refer to the digital resources and content that are the result of that digitization or digital creation process.

1.2. Motivating Use Cases

There are many different types of digitized or digital compound objects; ancient scrolls, paintings, letters, books, newspapers, films, operas, albums, field recordings and computer generated animations. Many of them bear the written or spoken word, sometimes difficult to read or hear either due to the decay of the physical object or lack of understanding of the script or language. These use cases are described in a separate document (/api/presentation/usecases/).

Collectively, the use cases require a model in which one can characterize the object (via the *Manifest* resource) and the individual views of the object (*Canvas* resources). Each view may have images, audio, video and other content resources associated with it (*Content* resources) to allow the view to be rendered to the user appropriately. An object may also have sections; for example, a book may have chapters of several pages, or a play might be divided into acts and scenes (*Range* resources) and there may be groups of objects (*Collection* resources). These resource types, along with their properties, make up the IIIF (International Image Interoperability Framework) Presentation API (Application Programming Interface).

1.3. Terminology

This specification uses the following terms:

- **embedded**: When a resource (A) is embedded within another resource (B), the complete JSON (JavaScript Object Notation) representation of resource A is present within the JSON (JavaScript Object Notation) representation of resource B, and dereferencing the URI of resource A will not result in additional information. Example: Canvas A is embedded in Manifest B.
- **referenced**: When a resource (A) is referenced from another resource (B), an incomplete JSON (JavaScript Object Notation) representation of resource A is present within the JSON (JavaScript Object Notation) representation of resource B, and dereferencing the URI of resource A will result in additional information. Typically, the `id`, `type`, and `label` properties of resource A will be included. Example: Manifest A is referenced from Collection B.
- **HTTP (Hypertext Transfer Protocol)(S)**: The HTTP (Hypertext Transfer Protocol) or HTTPS (Hypertext Transfer Protocol (Secure)) URI scheme and internet protocol.

The terms `array`, `JSON object`, `number`, `string`, `true`, `false`, `boolean` and `null` in this document are to be interpreted as defined by the Javascript Object Notation (JSON (JavaScript Object Notation)) (https://tools.ietf.org/html/rfc8259) specification.

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC (Request for Comments) 2119 (http://tools.ietf.org/html/rfc2119).

2. Resource Type Overview

This section provides an overview of the resource types (or classes) that are used in the specification. They are each presented in more detail in Section 5 (/api/presentation/3.0/#resource-structure).

2.1. Defined Types

This specification defines the following resource types:

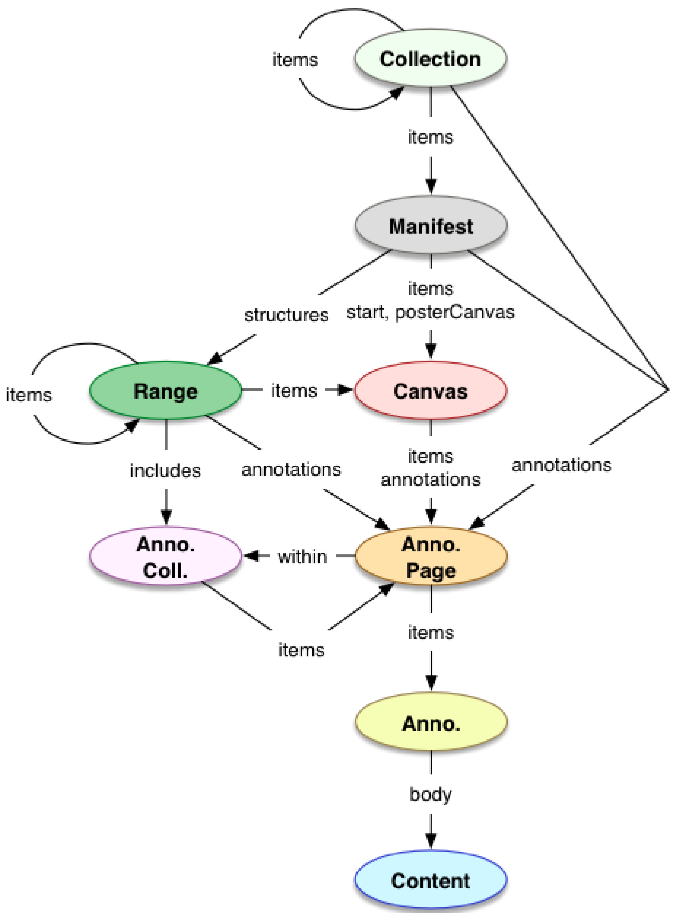
Collection
An ordered list of Manifests, and/or further Collections. Collections allow easy navigation among the Manifests in a hierarchical structure, potentially each with its own descriptive information. Collections might be used to model dynamic result sets from a search, fixed sets of related resources, or other groupings of Manifests for presentation.

Manifest
The overall description of the structure and properties of the digital representation of an object. It carries information needed for the viewer to present the content to the user, such as a title and other descriptive information about the object or the intellectual work that it conveys. Each Manifest describes how to present a single object such as a book, a statue or

a music album.

Canvas
A virtual container that represents a particular view of the object and has content resources associated with it or with parts of it. The Canvas provides a frame of reference for the layout of the content, both spatially and temporally. The concept of a Canvas is borrowed from standards like PDF and HTML (HyperText Markup Language), or applications like Photoshop and PowerPoint, where an initially blank display surface has images, video, text and other content "painted" on to it by Annotations, collected in Annotation Pages.

Range
An ordered list of Canvases, and/or further Ranges. Ranges allow Canvases, or parts thereof, to be grouped together in some way. This could be for content-based reasons, such as might be described in a table of contents or the set of scenes in a play. Equally, physical features might be important such as page gatherings in an early book, or when recorded music is split across different physical carriers such as two CDs.



2.2. Additional Types

This specification makes use of the following types, defined in the Web Annotation Data Model (http://w3.org/TR/annotation-model/) specification:

Annotation Page
An ordered list of Annotations that is typically associated with a Canvas but may be referenced from other resource types as well. Annotation Pages collect and order lists of Annotations, which in turn provide commentary about a resource or content that is part of a Canvas.

Annotation
Annotations associate Content resources with Canvases. The same mechanism is used for the visible and/or audible resources as is used for transcriptions, commentary, tags and other content. This provides a single, unified method for aligning information, and provides a standards-based framework for distinguishing parts of resources and parts of Canvases. As Annotations can be added later, it promotes a distributed system in which publishers can align their content with the descriptions created by others.

Content
Content resources such as images, audio, video or text that are associated with a Canvas via Annotations, or provide a rendering of any resource.

Annotation Collection
An ordered list of Annotation Pages. Annotation Collections allow higher level groupings of Annotations to be recorded. For example, all of the English translation Annotations of a medieval French document could be kept separate from the transcription or an edition in modern French, or the director's commentary on a film can be separated from the script.

3. Resource Properties

Most of the properties defined by this specification may be associated with any of the resource types described above, and may have more than one value. The property relates to the resource that it is associated with, so the `label` property on a Manifest is the human readable label of the Manifest, whereas the same `label` property on a Canvas is the human readable label for that particular view. In the descriptions of the properties, the resource that the property is associated with is called "this resource".

The requirements for which classes have which properties are summarized in Appendix A (/api/presentation/3.0/#a-summary-of-metadata-requirements).

Other properties are allowed, either via local extensions or those endorsed by the IIIF (International Image Interoperability Framework) community. If a client discovers properties that it does not understand, then it MUST ignore them. See the Linked Data Context and Extensions section for more information about extensions.

This section also defines processing requirements for clients for each of the combinations of class and property. These requirements are for general purpose client implementations that are intended to be used to render the entire resource to the user, and not necessarily for consuming applications with specialized use or individual component implementations that might be used to construct a client. The inclusion of these requirements gives publishers a baseline expectation for how they can expect implmentations advertised as compliant with this specification to behave when processing their content.

3.1. Descriptive Properties

These properties describe or represent the resource they are associated with ("this resource"), and are typically rendered to the user.

label

A human readable label, name or title for this resource. The `label` property is intended to be displayed as a short, textual surrogate for the resource if a human needs to make a distinction between it and similar resources, for example between objects, pages, or options for a choice of images to display. The `label` property can be fully internationalized, and each language can have multiple values. This pattern is described in more detail in the languages (/api/presentation/3.0/#language-of-property-values) section.

The value of the property MUST be a JSON (JavaScript Object Notation) object, as described in the languages (/api/presentation/3.0/#language-of-property-values) section.

- A Collection MUST have the `label` property with at least one entry.
  Clients MUST render `label` on a Collection.
- A Manifest MUST have the `label` property with at least one entry.
  Clients MUST render `label` on a Manifest.
- A Canvas SHOULD have the `label` property with at least one entry.
  Clients MUST render `label` on a Canvas, and SHOULD generate a `label` for Canvases that do not have them.
- A content resource MAY have the `label` property with at least one entry. If there is a Choice of content resource for the same Canvas, then they SHOULD each have at least the `label` property with at least one entry.
  Clients MAY render `label` on content resources, and SHOULD render them when part of a Choice.
- A Range SHOULD have the `label` property with at least one entry.
  Clients MUST render `label` on a Range.
- An Annotation Collection SHOULD have the `label` property with at least one entry.
  Clients SHOULD render `label` on an Annotation Collection.
- Other resource types MAY have the `label` property with at least one entry.
  Clients MAY render `label` on other resource types.

```
{ "label": { "en": [ "Example Object Title" ] } }
```

summary

A short textual summary of this resource, intended to be conveyed to the user when the `metadata` pairs for the resource are not being displayed. This could be used as a snippet for item level search results, for limited screen real-estate environments, or as an alternative user interface when the `metadata` property is not currently being rendered. The `summary` property follows the same pattern as the `label` property described above.

The value of the property MUST be a JSON (JavaScript Object Notation) object, as described in the languages (/api/presentation/3.0/#language-of-property-values) section.

- A Collection SHOULD have the `summary` property with at least one entry.
  Clients SHOULD render `summary` on a Collection.
- A Manifest SHOULD have the `summmary` property with at least one entry.
  Clients SHOULD render `summary` on a Manifest.
- A Canvas MAY have the `summary` property with at least one entry.
  Clients SHOULD render `summary` on a Canvas.
- Other resource types MAY have the `summary` property with at least one entry.
  Clients MAY render `summary` on other resource types.

```
{ "summary": { "en": [ "This is a summary of the object." ] } }
```

metadata

An ordered list of descriptive entries to be displayed to the user when they interact with this resource, given as pairs of human readable `label` and `value` entries. There are no semantics conveyed by this information, only strings to present to the user when interacting with this resource. A pair might be used to convey information about the creation of the object, a physical description, ownership information, and for many other use cases.

The value of the `metadata` property MUST be an array of JSON (JavaScript Object Notation) objects, where each item in the array has both `label` and `value` properties. The values of both `label` and `value` MUST be JSON (JavaScript Object Notation) objects, as described in the languages (/api/presentation/3.0/#language-of-property-values) section.

- A Collection SHOULD have the `metadata` property with at least one item.
  Clients MUST render `metadata` on a Collection.
- A Manifest SHOULD have the `metadata` property with at least one item.
  Clients MUST render `metadata` on a Manifest.
- A Canvas MAY have the `metadata` property with at least one item.
  Clients SHOULD render `metadata` on a Canvas.

- Other resource types MAY have the `metadata` property with at least one item.
  Clients MAY render `metadata` on other resource types.

Clients SHOULD display the pairs in the order provided. Clients SHOULD NOT use `metadata` for indexing and discovery purposes, as there are intentionally no consistent semantics. Clients SHOULD expect to encounter long texts in the `value` property, and render them appropriately, such as with an expand button, or in a tabbed interface.

```
{
  "metadata": [
    {
      "label": { "en": [ "Creator" ] },
      "value": { "en": [ "Anne Artist (1776-1824)" ] }
    }
  ]
}
```

requiredStatement

Text that MUST be displayed when this resource is displayed or used. For example, the `requiredStatement` property could be used to present copyright or ownership statements, an acknowledgement of the owning and/or publishing institution, or any other text that the organization deems critical to display to the user.

Given the wide variation of potential client user interfaces, it will not always be possible to display this statement to the user in the client's initial state. If initially hidden, clients MUST make the method of revealing it as obvious as possible.

The value of the property MUST be a JSON (JavaScript Object Notation) object, that has the `label` and `value` properties, in the same way as the `metadata` property items. The values of both `label` and `value` MUST be JSON (JavaScript Object Notation) objects, as described in the languages (/api/presentation/3.0/#language-of-property-values) section.

- Any resource type MAY have the `requiredStatement` property.
  Clients MUST render `requiredStatement` on every resource type.

```
{
  "requiredStatement": {
    "label": { "en": [ "Attribution" ] },
    "value": { "en": [ "Provided courtesy of Example Institution" ] }
  }
}
```

rights

A string that identifies a license or rights statement that applies to the content of this resource, such as the JSON (JavaScript Object Notation) of a Manifest or the pixels of an image. The value MUST be drawn from the set of Creative Commons (https://creativecommons.org/licenses/) licenses, the RightsStatements.org (http://rightsstatements.org/page/1.0/) rights statements, or those added via the extension mechanism. The inclusion of this property is informative, and for example could be used to display an icon representing the rights assertions.

If displaying rights information directly to the user is the desired interaction, or a publisher-defined label is needed, then it is RECOMMENDED to include the information using the `requiredStatement` property or in the `metadata` property.

The value MUST be a string. If the value is drawn from Creative Commons or RightsStatements.org, then the string MUST be a URI defined by that specification.

- Any resource type MAY have the `rights` property.
  Clients MAY render `rights` on any resource type.

```
{ "rights": "https://creativecommons.org/licenses/by/4.0/" }
```

thumbnail

An external content resource that represents this resource, such as a small image or short audio clip. It is RECOMMENDED that a IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) (/api/image/2.1/) service be available for images to enable manipulations such as resizing. The same resource MAY have multiple thumbnails with the same or different `type` and `format`.

The value MUST be an array of JSON (JavaScript Object Notation) objects, where each item in the array has an `id` property and SHOULD have at least one of the `type` and `format` properties.

- A Collection SHOULD have the `thumbnail` property with at least one item.
  Clients SHOULD render `thumbnail` on a Collection.
- A Manifest SHOULD have the `thumbnail` property with at least one item.
  Clients SHOULD render `thumbnail` on a Manifest.
- A Canvas MAY have the `thumbnail` property with at least one item. A Canvas SHOULD have the `thumbnail` property if there are multiple resources that make up the representation.
  Clients SHOULD render `thumbnail` on a Canvas.

- A content resource MAY have the `thumbnail` property with at least one item. Content resources SHOULD have the `thumbnail` property with at least one item if it is an option in a Choice of resources.
  Clients SHOULD render `thumbnail` on a content resource.
- Other resource types MAY have the `thumbnail` property with at least one item.
  Clients MAY render `thumbnail` on other resource types.

```
{ "thumbnail": [ { "id": "https://example.org/img/thumb.jpg", "type": "Image" } ] }
```

navDate
A date that the client can use for navigation purposes when presenting this resource to the user in a time-based user interface, such as a calendar or timeline. More descriptive date ranges, intended for display directly to the user, SHOULD be included in the `metadata` property for human consumption.

The value MUST be an `xsd:dateTime` literal (https://www.w3.org/TR/xmlschema11-2/#dateTime). The value MUST have a timezone, and SHOULD be given in UTC with the `Z` timezone indicator but MAY also be given as an offset of the form `+hh:mm`. In the situation where a timezone is not given, clients SHOULD assume it to be UTC.

- A Collection, Manifest or Range MAY have the `navDate` property.
  Clients MAY render `navDate` on Collections, Manifests and Ranges.
- Other resource types MUST NOT have the `navDate` property.
  Clients SHOULD ignore `navDate` on other resource types.

```
{ "navDate": "2010-01-01T00:00:00Z" }
```

posterCanvas
One or more Canvases providing content associated with the object represented by this resource but not part of that object. Examples include an image to show while a duration-only Canvas is playing audio; images, text and sound standing in for video content before the user initiates playback; or a film poster to attract user attention. The content provided by `posterCanvas` differs from a thumbnail: a client might use `thumbnail` to summarise and navigate multiple resources, then show content from `posterCanvas` as part of the presentation of a single resource. A poster Canvas can have different dimensions to those of the Canvas(es) of this resource.

Clients MAY display the content of a linked poster Canvas when presenting the resource, and if more than one is available MAY choose the one most suited to the client user interface. When more than one Canvas is available, for example if `posterCanvas` is provided for the currently selected Range and the current Manifest, and the client is able to show a poster Canvas, the client SHOULD pick the Canvas most specific to the content. Publishers SHOULD NOT assume that `posterCanvas` content will be seen in all clients. Clients SHOULD take care to avoid conflicts between time-based media in the `posterCanvas` and the content of the resource it is associated with.

The value MUST be a JSON (JavaScript Object Notation) object with the `id` and `type` properties, and MAY have other properties of Canvases.

- A Collection MAY have the `posterCanvas` property with at least one item.
  Clients MAY render `posterCanvas` on a Collection.
- A Manifest MAY have the `posterCanvas` property with at least one item.
  Clients MAY render `posterCanvas` on a Manifest.
- A Canvas MAY have the `posterCanvas` property with at least one item.
  Clients MAY render `posterCanvas` on a Canvas.
- A Range MAY have the `posterCanvas` property with at least one item.
  Clients MAY render `posterCanvas` on a Range.
- Other resource types MUST NOT have the `posterCanvas` property.
  Clients SHOULD ignore `posterCanvas` on other resource types.

```
{
  "posterCanvas": {
    "id": "https://example.org/iiif/1/canvas/poster",
    "type": "Canvas",
    "height": 1400,
    "width": 1200
    // ...
  }
}
```

3.2. Technical Properties

These properties describe technical features of the resources, and are typically processed by the client to understand how to render the resource.

id

The URI that identifies this resource. If this resource is only available embedded (see the terminology section (/api/presentation/3.0/#terminology) for an explanation of “embedded”) within another resource, such as a Range within a Manifest, then the URI MAY be the URI of the encapsulating resource with a unique fragment on the end. This is not true for Canvases, which MUST have their own URI without a fragment.

The value MUST be a string, and the value MUST be an HTTP (Hypertext Transfer Protocol)(S) URI for resources defined in this specification. If this resource is retrievable via HTTP (Hypertext Transfer Protocol)(S), then the URI MUST be the URI at which it is published. External resources, such as profiles, MAY have non-HTTP (Hypertext Transfer Protocol)(S) URIs defined by other communities.

- A Collection MUST have the `id` property.
  Clients SHOULD render `id` on a Collection.
- A Manifest MUST have the `id` property.
  Clients SHOULD render `id` on a Manifest.
- A Canvas MUST have the `id` property.
  Clients SHOULD render `id` on a Canvas.
- A content resource MUST have the `id` property.
  Clients MAY render `id` on content resources.
- A Range MUST have the `id` property.
  Clients MAY render `id` on a Range.
- An Annotation Collection MUST have the `id` property.
  Clients MAY render `id` on an Annotation Collection.
- An Annotation Page SHOULD have the `id` property.
  Clients MAY render `id` on an Annotation Page.
- An Annotation MUST have the `id` property.
  Clients MAY render `id` on an Annotation.

```
{ "id": "https://example.org/iiif/1/manifest" }
```

type
The type or class of this resource. For types defined by this specification, the value of `type` will be described in the sections below describing the individual classes. For external resources, the type is drawn from other specifications. Recommendations for basic types such as image, text or audio are given in the table below.

The value MUST be a string.

- All resource types MUST have the `type` property.
  Clients MUST process, and MAY render, `type` on any resource type.

Class	Description
`Application`	Software intended to be executed
`Dataset`	Data not intended to be rendered to humans directly
`Image`	Two dimensional visual resources primarily intended to be seen, such as might be rendered with an <img> HTML (HyperText Markup Language) tag
`Sound`	Auditory resources primarily intended to be heard, such as might be rendered with an <audio> HTML (HyperText Markup Language) tag
`Text`	Resources primarily intended to be read
`Video`	Moving images, with or without accompanying audio, such as might be rendered with a <video> HTML (HyperText Markup Language) tag

```
{ "type": "Dataset" }
```

format
The specific media type (often called a MIME type) for this content resource, for example “image/jpeg”. This is important for distinguishing different formats of the same overall type of resource, such as distinguishing text in XML (eXtensible Markup Language) from plain text.

The value MUST be a string, and it SHOULD be the value of the `Content-Type` header returned when this resource is dereferenced.

- A content resource SHOULD have the `format` propety.
  Clients MAY render the `format` of any content resource.

- Other resource types MUST NOT have the `format` property.
  Clients SHOULD ignore `format` on other resource types.

Note that this is different to the `formats` property in the Image API (Application Programming Interface) (/api/image/2.1/), which gives the extension to use within that API (Application Programming Interface). It would be inappropriate to use in this case, as `format` can be used with any content resource, not just images.

```
{ "format": "application/xml" }
```

language
The language or languages used in the content of this external resource. This property is already available from the Web Annotation model for content resources that are the body or target of an Annotation, however it MAY also be used for resources referenced (see the terminology section (/api/presentation/3.0/#terminology) for an explanation of "referenced") from `homepage`, `rendering`, `rights`, and `partOf`.

The value MUST be an array of strings. Each item in the array MUST be a valid language code, as described in the languages section (/api/presentation/3.0/#language-of-property-values).

- An external resource SHOULD have the `language` property with at least one item.
  Clients SHOULD process the `language` of external resources.
- Other resource types MUST NOT have the `language` property.
  Clients SHOULD ignore `language` on other resource types.

```
{ "language": [ "en" ] }
```

profile
A schema or named set of functionality available from this resource. The profile can further clarify the `type` and/or `format` of an external resource or service, allowing clients to customize their handling of this resource.

The value MUST be a string, either taken from the service registry (/api/annex/registry/services/) or a URI.

- Resources referenced by the `seeAlso` or `service` properties SHOULD have the `profile` property. Clients SHOULD process the `profile` of a service or external resource.
- Other resource types MAY have the `profile` property. Clients MAY process the `profile` of other resource types.

```
{ "profile": "https://example.org/profile/statuary" }
```

height
The height of this Canvas or external content resource. For content resources, the value is in pixels. For Canvases, the value does not have a unit. In combination with the width, it conveys an aspect ratio for the space in which content resources are located.

The value MUST be a positive integer.

- A Canvas MAY have the `height` property. If it has a `height`, it MUST also have a `width`.
  Clients MUST process `height` on a Canvas.
- Content resources MAY have the `height` property, with the value given in pixels, if appropriate.
  Clients SHOULD process `height` on content resources.
- Other resource types MUST NOT have the `height` property.
  Clients SHOULD ignore `height` on other resource types.

```
{ "height": 1800 }
```

width
The width of this Canvas or external content resource. For content resources, the value is in pixels. For Canvases, the value does not have a unit. In combination with the height, it conveys an aspect ratio for the space in which content resources are located.

The value MUST be a positive integer.

- A Canvas MAY have the `width` property. If it has a `width`, it MUST also have a `height`.
  Clients MUST process `width` on a Canvas.
- Content resources MAY have the `width` property, with the value given in pixels, if appropriate.
  Clients SHOULD process `width` on content resources.
- Other resource types MUST NOT have the `width` property.
  Clients SHOULD ignore `width` on other resource types.

```
{ "width": 1200 }
```

duration

The duration of this Canvas or external content resource, given in seconds.

The value MUST be a positive floating point number.

- A Canvas MAY have the `duration` property.
  Clients MUST process `duration` on a Canvas.
- Content resources MAY have the `duration` property.
  Clients SHOULD process `duration` on content resources.
- Other resource types MUST NOT have a `duration`.
  Clients SHOULD ignore `duration` on other resource types.

```
{ "duration": 125.0 }
```

viewingDirection
The direction that a set of Canvases SHOULD be displayed to the user. This specification defines four direction values in the table below. Others may be defined externally as an extension.

The value MUST be a string.

- A Collection MAY have the `viewingDirection` property.
  Clients SHOULD process `viewingDirection` on a Collection.
- A Manifest MAY have the `viewingDirection` property.
  Clients SHOULD process `viewingDirection` on a Manifest.
- A Range MAY have the `viewingDirection` property.
  Clients MAY process `viewingDirection` on a Range.
- Other resource types MUST NOT have the `viewingDirection` property.
  Clients SHOULD ignore `viewingDirection` on other resource types.

Value	Description
`left-to-right`	The object is displayed from left to right. The default if not specified.
`right-to-left`	The object is displayed from right to left.
`top-to-bottom`	The object is displayed from the top to the bottom.
`bottom-to-top`	The object is displayed from the bottom to the top.

```
{ "viewingDirection": "left-to-right" }
```

behavior
A set of user experience features that the publisher of the content would prefer the client to use when presenting this resource. This specification defines the values specified in the table below. Others may be defined externally as an extension.

The value MUST be an array of strings.

- Any resource type MAY have the `behavior` property with at least one item.
  Clients SHOULD process `behavior` on any resource type.

Value	Description
`auto-advance`	Valid on Collections, Manifests, Canvases, and Ranges that include or are Canvases with at least the `duration` dimension. When the client reaches the end of a Canvas with a duration dimension that has (or is within a resource that has) this `behavior`, it SHOULD immediately proceed to the next Canvas and render it. If there is no subsequent Canvas in the current context, then this `behavior` should be ignored. When applied to a Collection, the client should treat the first Canvas of the next Manifest as following the last Canvas of the previous Manifest, respecting any `start` property specified.
`continuous`	Valid on Manifests and Ranges, which include Canvases that have at least `height` and `width` dimensions. Canvases included in resources with this `behavior` are partial views and an appropriate rendering might display all of the Canvases virtually stitched together, such as a long scroll split into sections. This `behavior` has no implication for audio resources. The `viewingDirection` of the Manifest will determine the appropriate arrangement of the Canvases.
`facing-pages`	Valid only on Canvases, where the Canvas has at least `height` and `width` dimensions. Canvases with this `behavior`, in a Manifest with the “paged” `behavior`, MUST be displayed by themselves, as they depict both parts of the opening. If all of the Canvases are like this, then page turning is not possible, so simply use “individuals” instead.
`individuals`	Valid on Collections, Manifests, and Ranges. For Collections with this `behavior`, each of the included Manifests are distinct objects. For Manifest, and Range, the included Canvases are distinct views, and SHOULD NOT be presented in a page-turning interface. This is the default `behavior` if none are specified.
`multi-part`	Valid only on Collections. Collections with this `behavior` consist of multiple Manifests that each form part of a logical whole, such as multi-volume books or a set of journal issues. Clients might render the Collection as a table of contents, rather than with thumbnails.
`no-nav`	Valid only on Ranges. Ranges with this `behavior` MUST NOT be displayed to the user in a navigation hierarchy. This allows for Ranges to be present that capture unnamed regions with no interesting content, such as the set of blank pages at the beginning of a book, or dead air between parts of a performance, that are still part of the Manifest but do not need to be navigated to directly.
`non-paged`	Valid only on Canvases, where the Canvas has at least `height` and `width` dimensions. Canvases with this `behavior` MUST NOT be presented in a page turning interface, and MUST be skipped over when determining the page order. This `behavior` MUST be ignored if the current Manifest does not have the “paged” `behavior`.
`hidden`	Valid on Annotation Collections, Annotation Pages, Annotations, Specific Resources and Choices. If this `behavior` is provided, then the client SHOULD NOT render the resource by default, but allow the user to turn it on and off.
`paged`	Valid on Manifests and Ranges, which include Canvases that have at least `height` and `width` dimensions. Canvases included in resources with this `behavior` represent pages in a bound volume, and SHOULD be presented in a page-turning interface if one is available. The first canvas is a single view (the first recto) and thus the second canvas likely represents the back of the object in the first canvas. If this is not the case, see the “non-paged” `behavior`.
`repeat`	Valid on Collections and Manifests, that include Canvases with at least the `duration` dimension. When the client reaches the end of the duration of the final Canvas in the resource, and the “auto-advance” `behavior` is also in effect, then the client SHOULD return to the first Canvas in the resource with the “repeat” `behavior` and start playing again. If the “auto-advance” `behavior` is not in effect, then the client SHOULD render a navigation control for the user to manually return to the first Canvas.
`sequence`	Valid only on Ranges, where the Range is referenced in the `structures` property of a Manifest. Ranges with this `behavior` represent different orderings of the Canvases listed in the `items` property of the Manifest, and user interfaces that interact with this order SHOULD use the order within the selected Range, rather than the default order of `items`.
`thumbnail-nav`	Valid only on Ranges. Ranges with this `behavior` MAY be used by the client to present an alternative navigation or overview based on thumbnails, such as regular keyframes along a timeline for a video, or sections of a long scroll. Clients SHOULD NOT use them to generate a conventional table of contents. Child Ranges of a Range with this `behavior` MUST have a suitable `thumbnail` property.
`together`	Valid only on Collections. A client SHOULD present all of the child Manifests to the user at once in a separate viewing area with its own controls. Clients SHOULD catch attempts to create too many viewing areas. The “together” value SHOULD NOT be interpreted as applying to the members any child resources.
`unordered`	Valid on Manifests and Ranges. The order of Canvases within this resource have no inherent order, and user interfaces SHOULD avoid implying to the user that there is such an order.

```
{ "behavior": [ "auto-advance", "individuals" ] }
```

timeMode

A mode associated with an Annotation that is to be applied to the rendering of any time-based media, or otherwise could be considered to have a duration, used as a body resource of that Annotation. Note that the association of `timeMode` with the Annotation means that different resources in the body cannot have different values. This specification defines the values specified in the table below. Others may be defined externally as an extension.

The value MUST be a string.

- An Annotation MAY have the `timeMode` property.
  Clients SHOULD process `timeMode` on an Annotation.

Value	Description
`trim`	(default, if not supplied) If the content resource has a longer duration than the duration of portion of the Canvas it is associated with, then at the end of the Canvas's duration, the playback of the content resource MUST also end. If the content resource has a shorter duration than the duration of the portion of the Canvas it is associated with, then, for video resources, the last frame SHOULD persist on-screen until the end of the Canvas portion's duration. For example, a video of 120 seconds annotated to a Canvas with a duration of 100 seconds would play only the first 100 seconds and drop the last 20 second.
`scale`	Fit the duration of content resource to the duration of the portion of the Canvas it is associated with by scaling. For example, a video of 120 seconds annotated to a Canvas with a duration of 60 seconds would be played at double-speed.
`loop`	If the content resource is shorter than the `duration` of the Canvas, it MUST be repeated to fill the entire duration. Resources longer than the `duration` MUST be trimmed as described above. For example, if a 20 second duration audio stream is annotated onto a Canvas with duration 30 seconds, it will be played one and a half times.

```
{ "timeMode": "trim" }
```

3.3. Linking Properties

These properties are references or links between resources, and split into external references where the linked object is outside of the IIIF (International Image Interoperability Framework) space, and internal references where the linked object is a IIIF (International Image Interoperability Framework) resource. Clients typically create a link to the resource that is able to be activated by the user, or interact directly with the linked resource to improve the user's experience.

3.3.1. External Links

homepage
A link to an external web page that primarily describes the real world object that this resource represents. The web page is usually published by the organization responsible for the real world object, and might be generated by a content management system or other cataloging system. The external resource MUST be able to be displayed directly to the user. External resources that are related, but not home pages, MUST instead be added into the `metadata` property, with an appropriate `label` or `value` to describe the relationship.

The value MUST be a JSON (JavaScript Object Notation) object, which MUST have the `id`, `type`, and `label` properties, and SHOULD have a `format` property.

- Any resource type MAY have the `homepage` property.
  Clients SHOULD render `homepage` on a Collection, Manifest or Canvas, and MAY render `homepage` on other resource types.

```
{
  "homepage": {
    "id": "https://example.com/info/",
    "type": "Text",
    "label": { "en": [ "Homepage for Example Object" ] },
    "format": "text/html"
  }
}
```

logo
A small external image resource that represents an individual or organization associated with this resource. This could be the logo of the owning or hosting institution. The logo MUST be clearly rendered when the resource is displayed or used, without cropping, rotating or otherwise distorting the image. It is RECOMMENDED that a IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) (/api/image/2.1/) service be available for this image for other manipulations such as resizing.

The value MUST be an array of JSON (JavaScript Object Notation) objects, each of which MUST have an `id` and SHOULD have at least one of `type` and `format`.

- Any resource type MAY have the `logo` property with at least one item.
  Clients MUST render `logo` on every resource type.

```
{ "logo": [ { "id": "https://example.org/img/logo.jpg", "type": "Image" } ] }
```

rendering
A link to an external resource that is an alternative, non-IIIF (International Image Interoperability Framework) representation of this resource. Such representations typically cannot be painted onto a single Canvas, as they either include too many views, have incompatible dimensions, or are composite resources requiring additional rendering functionality. The external resource MUST be able to be displayed directly to a human user, although the presentation may be outside of the IIIF (International Image Interoperability Framework) client. The external resource MUST NOT have a splash page or other interstitial resource that mediates access to it. If access control is required, then the IIIF (International Image Interoperability Framework) Authentication API (Application Programming Interface) (/api/auth/) is RECOMMENDED. Examples include a rendering of a book as a PDF or EPUB, a slide deck with images of a building, or a 3D model of a statue. External resources that are not representations of this resource but are related in some other way MUST be referenced in the `metadata` property, or linked to a particular Canvas segment with an Annotation; a home page or preferred web view of the resource MUST be referenced in the `homepage` property.

The value MUST be an array of JSON (JavaScript Object Notation) objects. Each item MUST have the `id`, `type` and `label` properties, and SHOULD have a `format` property.

- Any resource type MAY have the `rendering` property with at least one item.
  Clients SHOULD render `rendering` on a Collection, Manifest or Canvas, and MAY render `rendering` on other resource types.

```
{
  "rendering": [
    {
      "id": "https://example.org/1.pdf",
      "type": "Text",
      "label": { "en": [ "PDF Rendering of Book" ] },
      "format": "application/pdf"
    }
  ]
}
```

service
A link to an external service that the client might interact with directly and gain additional information or functionality for using this resource, such as from an image to the base URI of an associated IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) (/api/image/2.1/) service. The service resource SHOULD have additional information associated with it in order to allow the client to determine how to make appropriate use of it. Please see the Service Registry (/api/annex/registry/services/) document for the details of currently known service types.

The value MUST be an array of JSON (JavaScript Object Notation) objects. Each object will have properties depending on the service's definition, but MUST have either the `id` or `@id` and `type` or `@type` properties. Each object SHOULD have a `profile` property.

- Any resource type MAY the `service` property with at least one item.
  Clients MAY process `service` on any resource type, and SHOULD process the IIIF (International Image Interoperability Framework) Image API (Application Programming Interface) service.

```
{
  "service": [
    {
      "id": "https://example.org/service",
      "type": "Service",
      "profile": "https://example.org/docs/service"
    }
  ]
}
```

For cross-version consistency, this specification defines the following values for the `type` or `@type` property for backwards compatibility with other IIIF (International Image Interoperability Framework) APIs. Future versions of these APIs will define their own types. These `type` values are necessary extensions for compatibility of the older versions.

Value	Specification
ImageService1	Image API (Application Programming Interface) version 1 (/api/image/1.1/)
ImageService2	Image API (Application Programming Interface) version 2 (/api/image/2.1/)
SearchService1	Search API (Application Programming Interface) version 1 (/api/search/1.0/)

Value	Specification
AutoCompleteService1	Search API (Application Programming Interface) version 1 (/api/search/1.0/#autocomplete)
AuthCookieService1	Authentication API (Application Programming Interface) version 1 (/api/auth/1.0/#access-cookie-service)
AuthTokenService1	Authentication API (Application Programming Interface) version 1 (/api/auth/1.0/#access-token-service)
AuthLogoutService1	Authentication API (Application Programming Interface) version 1 (/api/auth/1.0/#logout-service)

A reference from a version 3 Presentation API (Application Programming Interface) document to both version 2 and version 3 Image API (Application Programming Interface) endpoints would thus follow the pattern in the example below. API (Application Programming Interface) versions defined after Presentation API (Application Programming Interface) version 3.0 will follow the community best practices of `id` and `type`, but in the interim implementations MUST be prepared to recognize the older property names. Note that the `@context` key SHOULD not be present within the `service`, but instead included at the beginning of the document.

```
{
  "service": [
    {
      "@id": "https://example.org/iiif2/image1/identifier",
      "@type": "ImageService2",
      "profile": "http://iiif.io/api/image/2/level2.json"
    },
    {
      "id": "https://example.org/iiif3/image1/identifier",
      "type": "ImageService3",
      "profile": "level2"
    }
  ]
}
```

seeAlso

A link to an external, machine-readable resource that is related to this resource, such as an XML (eXtensible Markup Language) or RDF (Resource Description Framework) description. Properties of the external resource should be given to help the client select between multiple descriptions (if provided), and to make appropriate use of the document. If the relationship between the resource and the document needs to be more specific, then the document should include that relationship rather than the IIIF (International Image Interoperability Framework) resource. Other IIIF (International Image Interoperability Framework) resources, such as a related Manifest, are valid targets for `seeAlso`. The URI of the document MUST identify a single representation of the data in a particular format. For example, if the same data exists in JSON (JavaScript Object Notation) and XML (eXtensible Markup Language), then separate resources should be added for each representation, with distinct `id` and `format` properties.

The value MUST be an array of JSON (JavaScript Object Notation) objects. Each item MUST have the `id` and `type` properties, and SHOULD have the `label`, `format` and `profile` properties.

- Any resource type MAY have the `seeAlso` property with at least one item.
  Clients MAY process `seeAlso` on any resource type.

```
{
  "seeAlso": [
    {
      "id": "https://example.org/library/catalog/book1.xml",
      "type": "Dataset",
      "format": "text/xml",
      "profile": "https://example.org/profiles/bibliographic"
    }
  ]
}
```

3.4.2. Internal Links

partOf

A link to another resource that contains this resource, for example a link from a Manifest to a Collection that it is part of. When a client encounters the `partOf` property, it might retrieve the referenced resource, if it is not included in the current representation, in order to contribute to the processing of this resource. For example, if a Canvas is encountered as a stand alone resource, it might be part of a Manifest that includes other contextual information, or a Manifest might be part of a Collection, which would aid in navigation.

The value MUST be an array of JSON (JavaScript Object Notation) objects. Each item MUST have the `id` and `type` properties, and SHOULD have the `label` property.

- Any resource type MAY have the `partOf` property with at least one item
  Clients MAY render `partOf` on any resource type.

```
{ "partOf": [ { "id": "https://example.org/iiif/1", "type": "Manifest" } ] }
```

start
A link from this Manifest or Range, to a Canvas, or part of a Canvas, that is contained within it. The reference to part of a Canvas is handled in the same way that Ranges reference parts of Canvases. When processing this relationship, a client SHOULD advance to the specified Canvas, or specified segment of the Canvas, when beginning navigation through the Range. This allows the client to begin with the first Canvas that contains interesting content rather than requiring the user to manually navigate to find it.

The value MUST be a JSON (JavaScript Object Notation) object, which MUST have the `id` and `type` properties. The object MUST be either a Canvas (as in the first example below), or a Specific Resource with a Selector and a `source` property where the value is a Canvas (as in the second example below).

- A Manifest or Range MAY have the `start` property. Clients SHOULD process `start` on a Manifest or Range.
- Other resource types MUST NOT have the `start` property. Clients SHOULD ignore `start` on other resource types.

```
{ "start": { "id": "https://example.org/iiif/1/canvas/1", "type": "Canvas" } }
```

```
{
  "start": {
    "type": "SpecificResource",
    "source": "https://example.org/iif/1/canvas/1",
    "selector": {
      "type": "PointSelector",
      "t": 14.5
    }
  }
}
```

supplementary
A link from this Range to an Annotation Collection that includes the “supplementing” Annotations of content resources for the Range. Clients might use this to present additional content to the user from a different Canvas when interacting with the Range, or to jump to the next part of the Range within the same Canvas. For example, the Range might represent a newspaper article that spans non-sequential pages, and then uses the `supplementary` property to reference an Annotation Collection that consists of the Annotations that record the text, split into Annotation Pages per newspaper page.

The value MUST be a JSON (JavaScript Object Notation) object, which MUST have the `id` and `type` properties, and the `type` MUST be `AnnotationCollection`.

- A Range MAY have the `supplementary` property.
  Clients MAY process `supplementary` on a Range.
- Other resource types MUST NOT have the `supplementary` property.
  Clients SHOULD ignore `supplementary` on other resource types.

```
{ "supplementary": { "id": "https://example.org/iiif/1/annos/1", "type": "AnnotationCollection" } }
```

3.4. Structural Properties

These properties define the structure of the object being represented in IIIF (International Image Interoperability Framework) by allowing the inclusion of child resources within parents, such as a Canvas within a Manifest, or a Manifest within a Collection. The majority of cases use `items`, however there are two special cases for different sorts of structures.

items
Much of the functionality of the IIIF (International Image Interoperability Framework) Presentation API (Application Programming Interface) is simply recording the order in which child resources occur within a parent resource, such as Collections or Manifests within a parent Collection, or Canvases within a Manifest. All of these situations are covered with a single property, `items`.

The value MUST be an array of JSON (JavaScript Object Notation) objects. Each item MUST have the `id` and `type` properties. The items will be resources of different types, as described below.

- A Collection MUST have the `items` property. Each item MUST be either a Collection or a Manifest.
  Clients MUST process `items` on a Collection.
- A Manifest MUST have the `items` property with at least one item. Each item MUST be a Canvas.
  Clients MUST process `items` on a Manifest.
- A Canvas SHOULD have the `items` property with at least one item. Each item MUST be an Annotation Page.
  Clients MUST process `items` on a Canvas.

- An Annotation Page SHOULD have the `items` property with at least one item. Each item MUST be an Annotation. Clients MUST process `items` on an Annotation Page.
- A Range MUST have the `items` property with at least one item. Each item MUST be a Range, a Canvas or a Specific Resource where the source is a Canvas.
  Clients SHOULD process `items` on a Range.

```
{
  "items": [
    {
      "id": "https://example.org/iiif/manifest1",
      "type": "Manifest"
    },
    {
      "id": "https://example.org/iiif/collection1",
      "type": "Collection"
    },
    // ...
  ]
}
```

structures

The structure of an object represented as a Manifest can be described using a hierarchy of Ranges. Ranges can be used to describe the “table of contents” of the object or other structures that the user can interact with beyond the order given by the `items` property of the Manifest. The hierarchy is built by nesting the child Range resources in the `items` array of the higher level Range. The top level Ranges of these hierarchies are given in the `structures` property.

The value MUST be an array of JSON (JavaScript Object Notation) objects. Each item MUST have the `id` and `type` properties, and the `type` MUST be `Range`.

- A Manifest MAY have the `structures` property.
  Clients SHOULD process `structures` on a Manifest. The first hierarchy SHOULD be presented to the user by default, and further hierarchies SHOULD be able to be selected as alternative structures by the user.

```
{
  "structures": [
    {
      "id": "https://example.org/iiif/range/1",
      "type": "Range",
      "items": [ { ... } ]
    }
  ]
}
```

annotations

An ordered list of Annotation Pages that contain commentary or other Annotations about this resource, separate from the annotations that are used to paint content on to a Canvas. The `motivation` of the Annotations MUST NOT be “painting”, and the target of the Annotations MUST include this resource or part of it.

The value MUST be an array of JSON (JavaScript Object Notation) objects. Each item MUST have at least the `id` and `type` properties.

- A Collection, Manifest, Canvas, Range or content resource MAY have the `annotations` property with at least one item.
  Clients SHOULD process `annotations` on a Collection, Manifest, Canvas, Range or content resource.
- Other resource types MUST NOT have the `annotations` property. Clients SHOULD ignore `annotations` on other resource types.

```
{
  "annotations": [
    {
      "id": "https://example.org/iiif/annotationPage/1",
      "type": "AnnotationPage",
      "items": [ { ... } ]
    }
  ]
}
```

3.6. Values

Values for motivation

This specification defines two values for the Web Annotation property of `motivation`, or `purpose` when used on a Specific Resource or Textual Body.

While any resource MAY be the `target` of an Annotation, this specification defines only motivations for Annotations that target Canvases. These motivations allow clients to determine how the Annotation should be rendered, by distinguishing between Annotations that provide the content of the Canvas, from ones with externally defined motivations which are typically comments about the Canvas.

Additional motivations may be added to the Annotation to further clarify the intent, drawn from extensions or other sources. Clients MUST ignore motivation values that they do not understand. Other motivation values given in the Web Annotation specification SHOULD be used where appropriate, and examples are given in the Presentation API (Application Programming Interface) Cookbook (/api/annex/cookbook/).

Value	Description
`painting`	Resources associated with a Canvas by an Annotation with the “painting” motivation MUST be presented to the user as the representation of the Canvas. The content can be thought of as being “of” the Canvas. The use of this motivation with target resources other than Canvases is undefined. For example, an Annotation with the “painting” motivation, a body of an Image and the target of the Canvas is an instruction to present that Image as (part of) the visual representation of the Canvas. Similarly, a textual body is to be presented as (part of) the visual representation of the Canvas and not positioned in some other part of the user interface.
`supplementing`	Resources associated with a Canvas by an Annotation with the “supplementing” motivation MAY be presented to the user as part of the representation of the Canvas, or MAY be presented in a different part of the user interface. The content can be thought of as being “from” the Canvas. The use of this motivation with target resources other than Canvases is undefined. For example, an Annotation with the “supplementing” motivation, a body of an Image and the target of part of the Canvas is an instruction to present that Image to the user either in the Canvas’s rendering area or somewhere associated with it, and could be used to present an easier to read representation of a diagram. Similar, a textual body is to be presented either in the targeted region of the Canvas or otherwise associated with it, and might be a transcription of handwritten text or captions for what is being said in a Canvas with audio content.

4. JSON-LD (JSON for Linking Data) Considerations

This section describes features applicable to all of the Presentation API (Application Programming Interface) content. For the most part, these are features of the JSON-LD (JSON for Linking Data) specification that have particular uses within the API (Application Programming Interface) and recommendations about URIs to use.

4.1. Resource Representations

Resource descriptions SHOULD be embedded within the JSON (JavaScript Object Notation) description of parent resources, and MAY also be available via separate requests from the HTTP (Hypertext Transfer Protocol)(S) URI given in the resource’s `id` property. Links to resources MUST be given as a JSON (JavaScript Object Notation) object with the `id` property and at least one other property, typically either `type`, `format` or `profile` to give a hint as to what sort of resource is being referred to.

```
{
  "rendering": [
    {
      "id": "https://example.org/content/book.pdf",
      "type": "Text",
      "label": "Example Book (pdf)",
      "format": "application/pdf"
    }
  ]
}
```

4.2. Properties with Multiple Values

Any of the properties in the API (Application Programming Interface) that can have multiple values MUST always be given as an array of values, even if there is only a single item in that array.

```
{
  "thumbnail": [
    { "id": "https://example.org/images/thumb1.jpg", "type": "Image" }
  ]
}
```

4.3. Language of Property Values

Language MAY be associated with strings that are intended to be displayed to the user for the `label` and `summary` properties, plus the `label` and `value` properties of the `metadata` and `requiredStatement` objects.

The values of these properties MUST be JSON (JavaScript Object Notation) objects, with the keys being the RFC (Request for Comments) 5646 (http://tools.ietf.org/html/rfc5646) language code for the language, or if the language is either not known or the string does not have a language, then the key must be `"@none"`. The associated values MUST be arrays of strings, where each item is the content in the given language.

```
{
  "label": {
    "en": [
      "Whistler's Mother",
      "Arrangement in Grey and Black No. 1: The Artist's Mother"
    ],
    "fr": [
      "Arrangement en gris et noir no 1",
      "Portrait de la mère de l'artiste",
      "La Mère de Whistler"
    ],
    "@none": [ "Whistler (1871)" ]
  }
}
```

Note that RFC (Request for Comments) 5646 (http://tools.ietf.org/html/rfc5646) allows the script of the text to be included after a hyphen, such as `ar-latn`, and clients should be aware of this possibility.

In the case where multiple values are supplied, clients MUST use the following algorithm to determine which values to display to the user.

- If all of the values are associated with the `@none` key, the client MUST display all of those values.
- Else, the client should try to determine the user's language preferences, or failing that use some default language preferences. Then:
  - If any of the values have a language associated with them, the client MUST display all of the values associated with the language that best matches the language preference.
  - If all of the values have a language associated with them, and none match the language preference, the client MUST select a language and display all of the values associated with that language.
  - If some of the values have a language associated with them, but none match the language preference, the client MUST display all of the values that do not have a language associated with them.

Note that this does not apply to embedded textual bodies in Annotations, which use the Web Annotation pattern of `value` and `language` as separate properties.

4.4. HTML (HyperText Markup Language) Markup in Property Values

Minimal HTML (HyperText Markup Language) markup MAY be included in the `summary` property and the `value` property in the `metadata` and `requiredStatement` objects. It MUST NOT be used in `label` or other properties. This is included to allow Manifest creators to add links and simple formatting instructions to blocks of text. The content MUST be well-formed XML (eXtensible Markup Language) and therefore must be wrapped in an element such as `p` or `span`. There MUST NOT be whitespace on either side of the HTML (HyperText Markup Language) string, and thus the first character in the string MUST be a '<' character and the last character MUST be '>', allowing a consuming application to test whether the value is HTML (HyperText Markup Language) or plain text using these. To avoid a non-HTML (HyperText Markup Language) string matching this, it is RECOMMENDED that an additional whitespace character be added to the end of the value in situations where plain text happens to start and end this way.

In order to avoid HTML (HyperText Markup Language) or script injection attacks, clients MUST remove:

- Tags such as `script`, `style`, `object`, `form`, `input` and similar.
- All attributes other than `href` on the `a` tag, `src` and `alt` on the `img` tag.
- All `href` attributes that start with the string "javascript:"
- CData sections.
- XML (eXtensible Markup Language) Comments.
- Processing instructions.

Clients SHOULD allow only `a`, `b`, `br`, `i`, `img`, `p`, `small`, `span`, `sub` and `sup` tags. Clients MAY choose to remove any and all tags, therefore it SHOULD NOT be assumed that the formatting will always be rendered.

```
{ "summary": { "en-latn": [ "<p>Short <b>summary</b> of the resource.</p>" ] } }
```

4.5. Linked Data Context and Extensions

The top level resource in the response MUST have the `@context` property, and it SHOULD appear as the very first key/value pair of the JSON (JavaScript Object Notation) representation. This tells Linked Data processors how to interpret the document. The IIIF (International Image Interoperability Framework) Presentation API (Application Programming Interface) context, below, MUST occur once per response in the top-most resource, and thus MUST NOT appear within embedded resources. For example, when embedding a Canvas within a Manifest, the Canvas will not have the `@context` property.

The value of the `@context` property MUST be an array, and the **last** two values MUST be the Web Annotation context and the Presentation API (Application Programming Interface) context, in that order. Further contexts, such as those for local or registered extensions (/api/annex/registry/), MUST be added at the beginning of the array.

```
{
  "@context": [
    "http://www.w3.org/ns/anno.jsonld",
    "http://iiif.io/api/presentation/3/context.json"
  ]
}
```

Any additional properties beyond those defined in this specification or the Web Annotation Data Model SHOULD be mapped to RDF (Resource Description Framework) predicates using further context documents. These extensions SHOULD be added to the top level `@context` property, and MUST be added before the above contexts. The JSON-LD (JSON for Linking Data) 1.1 functionality of predicate specific context definitions, known as scoped contexts (https://json-ld.org/spec/latest/json-ld/#scoped-contexts), MUST be used to minimize cross-extension collisions. Extensions intended for community use SHOULD be registered in the extensions registry (/api/annex/registry/), but registration is not mandatory.

The JSON (JavaScript Object Notation) representation MUST NOT include the `@graph` key at the top level. This key might be created when serializing directly from RDF (Resource Description Framework) data using the JSON-LD (JSON for Linking Data) 1.0 compaction algorithm. Instead, JSON-LD (JSON for Linking Data) framing and/or custom code should be used to ensure the structure of the document is as defined by this specification.

5. Resource Structure

This section provides detailed description of the resource types used in this specification. Section 2 (/api/presentation/3.0/#resource-type-overview) provides an overview of the resource types and figures illustrating allowed relationships between them, and Appendix A (/api/presentation/3.0/#a-summary-of-metadata-requirements) provides summary tables of the property requirements.

5.1. Collection

Collections are used to list the Manifests available for viewing. Collections MAY include both other Collections and Manifests, in order to form a tree-structured hierarchy. Collections might align with the curated management of cultural heritage resources in sets, also called “collections”, but may have absolutely no such similarity.

The intended usage of Collections is to allow clients to:

- Load a pre-defined set of Manifests at initialization time.
- Receive a set of Manifests, such as search results, for rendering.
- Visualize lists or hierarchies of related Manifests.
- Provide navigation through a list or hierarchy of available Manifests.

Collections MAY be embedded inline within other Collections, such as when the Collection is used primarily to subdivide a larger one into more manageable pieces, however Manifests MUST NOT be embedded within Collections. An embedded Collection SHOULD also have its own URI from which the JSON (JavaScript Object Notation) description is available.

Manifests or Collections MAY be referenced from more than one Collection. For example, an institution might define four Collections: one for modern works, one for historical works, one for newspapers and one for books. The Manifest for a modern newspaper would then appear in both the modern Collection and the newspaper Collection. Alternatively, the institution may choose to have two separate newspaper Collections, and reference each as a sub-Collection of modern and historical.

Collections with an empty `items` property are allowed but discouraged. For example, if the user performs a search that matches no Manifests, then the server MAY return a Collection response with no Manifests.

An example Collection document:

```
{
  "@context": [
    "http://www.w3.org/ns/anno.jsonld",
    "http://iiif.io/api/presentation/3/context.json"
  ],
  "id": "https://example.org/iiif/collection/top",
  "type": "Collection",
  "label": { "en": [ "Collection for Example Organization" ] },
  "summary": { "en": [ "Short summary of the Collection" ] },
  "requiredStatement": {
    "label": { "en": [ "Attribution" ] },
    "value": { "en": [ "Provided by Example Organization" ] }
  },

  "items": [
    {
      "id": "https://example.org/iiif/1/manifest",
      "type": "Manifest",
      "label": { "en": "Example Manifest 1" }
    }
  ]
}
```

Note that while the Collection MAY reference Collections or Manifests from previous versions of the API (Application Programming Interface), the information included in this document MUST follow the current version requirements, not the requirements of the target document. This is in contrast to the requirements of `service`, as there is no way to distinguish a version 2 Manifest from a version 3 Manifest by its `type`.

5.2. Manifest

The Manifest resource typically represents a single object and any intellectual work or works embodied within that object. In particular it includes descriptive, rights and linking information for the object. It embeds the Canvases that should be rendered to the user as views of the object. The Manifest response contains sufficient information for the client to initialize itself and begin to display something quickly to the user.

The identifier in `id` MUST be able to be dereferenced to retrieve the JSON (JavaScript Object Notation) description of the Manifest, and thus MUST use the HTTP (Hypertext Transfer Protocol)(S) URI scheme.

The Manifest MUST have an `items` property, which is an array of JSON-LD (JSON for Linking Data) objects. Each object is a Canvas, described in the next section. It MAY also have a `structures` property listing one or more Ranges (/api/presentation/3.0/#range) which describe additional structure of the content, such as might be rendered as a table of contents. The Manifest MAY have an `annotations` property, which includes Annotation Page resources where the Annotations have the Manifest as their `target`. These will typically be comment style annotations, and MUST NOT have `painting` as their `motivation`.

```
{
  // Metadata about this manifest file
  "@context": [
    "http://www.w3.org/ns/anno.jsonld",
    "http://iiif.io/api/presentation/3/context.json"
  ],
  "id": "https://example.org/iiif/book1/manifest",
  "type": "Manifest",

  // Descriptive metadata about the object/work
  "label": { "en": [ "Book 1" ] },
  "metadata": [
    {
      "label": { "en": [ "Author" ] },
      "value": { "@none": [ "Anne Author" ] }
    },
    {
      "label": { "en": [ "Published" ] },
      "value": {
        "en": [ "Paris, circa 1400" ],
        "fr": [ "Paris, environ 1400" ]
      }
    },
    {
      "label": { "en": [ "Notes" ] },
      "value": {
        "en": [
          "Text of note 1",
          "Text of note 2"
        ]
      }
    },
    {
      "label": { "en": [ "Source" ] },
      "value": { "@none": [ "<span>From: <a href=\"https://example.org/db/1.html\">Some Collection</a>
</span>" ] }
    }
  ],
  "summary": { "en": [ "Book 1, written be Anne Author, published in Paris around 1400." ] },

  "thumbnail": [
    {
      "id": "https://example.org/images/book1-page1/full/80,100/0/default.jpg",
      "type": "Image",
      "service": [
        {
          "id": "https://example.org/images/book1-page1",
          "type": "ImageService3",
          "profile": "level1"
        }
      ]
    }
  ],

  // Presentation Information
  "viewingDirection": "right-to-left",
  "behavior": [ "paged" ],
  "navDate": "1856-01-01T00:00:00Z",

  // Rights Information
  "rights": [
    {
      "id":"https://example.org/license.html",
      "type": "Text",
      "language": "en",
      "format": "text/html"
    }
  ],
  "requiredStatement": {
    "label": { "en": [ "Attribution" ] },
    "value": { "en": [ "Provided by Example Organization" ] }
  },
  "logo": {
    "id": "https://example.org/logos/institution1.jpg",
    "type": "Image",
    "service": [
      {
        "id": "https://example.org/service/inst1",
        "type": "ImageService3",
        "profile": "level2"
      }
    ]
  },

  // Links
  "homepage": {
    "id": "https://example.org/info/book1/",
    "type": "Text",
    "label": { "en": [ "Home page for Book 1" ] },
    "format": "text/html"
  },
  "service": [
    {
      "id": "https://example.org/service/example",
      "type": "Service",
      "profile": "https://example.org/docs/example-service.html"
    }
  ],
  "seeAlso": [
    {
      "id": "https://example.org/library/catalog/book1.xml",
      "type": "Dataset",
      "format": "text/xml",
      "profile": "https://example.org/profiles/bibliographic"
    }
  ],
  "rendering": [
    {
      "id": "https://example.org/iiif/book1.pdf",
      "type": "Text",
      "label": { "en": [ "Download as PDF" ] },
      "format": "application/pdf"
    }
  ],
  "partOf": [
    {
      "id": "https://example.org/collections/books/",
      "type": "Collection"
    }
  ],
  "start": {
    "id": "https://example.org/iiif/book1/canvas/p2",
    "type": "Canvas"
  },

  // List of Canvases
  "items": [
    {
      "id": "https://example.org/iiif/book1/canvas/p1",
      "type": "Canvas",
      "label": { "@none": [ "p. 1" ] }
      // ...
    }
  ],

  // structure of the resource, described with Ranges
  "structures": [
    {
      "id": "https://example.org/iiif/book1/range/top",
      "type": "Range"
      // Ranges members should be included here
    }
    // Any additional top level Ranges can be included here
  ],

  // Commentary Annotations on the Manifest
  "annotations": [
    {
      "id": "https://example.org/iiif/book1/annotations/p1",
      "type": "AnnotationPage",
      "items": [
        // Annotations about the Manifest are included here
      ]
    }
  ]
}
```

5.3. Canvas

The Canvas represents an individual page or view and acts as a central point for assembling the different content resources that make up the display. Canvases MUST be identified by a URI and it MUST be an HTTP (Hypertext Transfer Protocol)(S) URI. The URI of the canvas MUST NOT contain a fragment (a # followed by further characters), as this would make it impossible to refer to a segment of the Canvas's area using the media fragment syntax (http://www.w3.org/TR/media-frags/) of `#xywh=` for spatial regions, and/or `#t=` for temporal segments. Canvases MAY be able to be dereferenced separately from the Manifest via their URIs as well as being embedded.

Every Canvas SHOULD have a `label` to display. If one is not provided, the client SHOULD automatically generate one for use based on the Canvas's position within the `items` property.

Content resources are associated with the Canvas via Web Annotations. Content that is to be rendered as part of the Canvas MUST be associated by an Annotation with the "painting" `motivation`. These Annotations are recorded in the `items` of one or more Annotation Pages, referred to in the `items` array of the Canvas. Annotations that do not have the "painting" `motivation` MUST NOT be in pages referenced in `items`, but instead in the `annotations` property.

Content that is derived from the Canvas, such as a transcription of text in an image or the words spoken in an audio representation, MUST be associated by an Annotation with the "supplementing" `motivation`. Annotations MAY have any other `motivation` as well. Thus content of any type may be associated with the Canvas via a "painting" annotation meaning the content is part of the Canvas, a "supplementing" annotation meaning the content is from the Canvas but not necessarily part of it, or an Annotation with another `motivation` meaning that it is somehow about the Canvas.

A Canvas MUST have a rectangular aspect ratio (described with the `height` and `width` properties) and/or a `duration` to provide an extent in time. These dimensions allow resources to be associated with specific regions of the Canvas, within the space and/or time extents provided. Content MUST NOT be associated with space or time outside of the Canvas's dimensions, such as at coordinates below 0,0, greater than the height or width, before 0 seconds, or after the duration. Content resources that have dimensions which are not defined for the Canvas MUST NOT be associated with that Canvas by an Annotation with the "painting" motiviation. For example, it is valid to use a "painting" Annotation to associate an Image (which has only height and width) with a Canvas that has all three dimensions, but it is an error to associate a Video resource (which has height, width and duration) with a Canvas that does not have all three dimensions. Such a resource SHOULD instead be referenced with the `rendering` property, or by Annotations with a `motivation` other than "painting" in the `annotations` property.

Parts of Canvases MAY be described using a Specific Resource with a Selector, following the patterns defined in the Web Annotation (http://w3.org/TR/annotation-model/) data model. The use of the `FragmentSelector` class is RECOMMENDED by that specification, as it allows for refinement by other Selectors and for consistency with use cases that cannot be represented using a URI fragment directly. Parts of Canvases can be referenced from Ranges, as the `body` or `target` of Annotations, or in the `start` property.

Parts of Canvases MAY also be identified by appending a fragment to the Canvas's URI, and these parts are still considered to be Canvases and have a `type` of "Canvas". Rectangular spatial parts of Canvases MAY also be described by appending an `xywh=` fragment to the end of the Canvas's URI. Similarly, temporal parts of Canvases MAY be described by appending a `t=` fragment to the end of the Canvas's URI. Spatial and temporal fragments MAY be combined, using an `&` character between them, and the temporal dimension SHOULD come first. It is an error to select a region using a dimension that is not defined by the Canvas, such as a temporal region of a Canvas that only has height and width dimensions.

Canvases MAY be treated as content resources for the purposes of annotating on to other Canvases. For example, a Canvas (Canvas A) with a video resource and Annotations representing subtitles or captions may be annotated on to another Canvas (Canvas B). This pattern maintains the correct spatial and temporal alignment of Canvas A's content relative to Canvas B's dimensions.

Renderers MUST scale content into the space represented by the Canvas, and SHOULD follow any `timeMode` value provided for time-based media. If the Canvas represents a view of a physical object, the spatial dimensions of the Canvas SHOULD be the same scale as that physical object, and content SHOULD represent only the object.

```
{
  // Metadata about this canvas
  "id": "https://example.org/iiif/book1/canvas/p1",
  "type": "Canvas",
  "label": { "@none": [ "p. 1" ] },
  "height": 1000,
  "width": 750,
  "duration": 180.0,

  "items": [
    {
      "id": "https://example.org/iiif/book1/page/p1/1",
      "type": "AnnotationPage",
      "items": [
        // Content Annotations on the Canvas are included here
      ]
    }
  ]
}
```

5.4. Range

Ranges are used to represent structure within an object beyond the default order of the Canvases in the `items` property of the Manifest, such as newspaper sections or articles, chapters within a book, or movements within a piece of music. Ranges can include Canvases, parts of Canvases, or other Ranges, creating a tree structure like a table of contents.

The intent of adding a Range to the Manifest is to allow the client to display a linear or hierarchical navigation interface to enable the user to quickly move through the object's content. Clients SHOULD present only Ranges with the `label` property and without the "no-nav" `behavior` to the user. Clients SHOULD NOT render Canvas labels as part of the navigation, and a Range that wraps the Canvas MUST be created if this is the desired presentation.

If there is no Range that has the `behavior` "sequence", and the Manifest does not have the `behavior` "unordered", then the client SHOULD treat the order of the Canvases in the Manifest's `items` array as the default order. If there is one Range with the `behavior` value "sequence", then the viewer MUST instead use this Range for the ordering. If there is more than one Range with the `behavior` value "sequence", for example a second Range to represent an alternative ordering of the pages of a manuscript, the first Range SHOULD be used as the default and the others SHOULD be able to be selected. Ranges with the `behavior` value "sequence" MUST be directly within the `structures` property of the Manifest, and MUST NOT be embedded or referenced within other Ranges. These Ranges may have limited hierarchical nesting, but clients are not expected to traverse very deep structures in determining the default order. If this Range includes parts of Canvases, then these parts are the content to render by default and would generate separate entries in a navigation display. This allows for the Canvas to include content outside of the default view, such as a color bar or ruler.

Ranges MUST have URIs and they SHOULD be HTTP (Hypertext Transfer Protocol)(S) URIs. Top level Ranges are embedded or externally referenced within the Manifest in a `structures` property. These top level Ranges then embed or reference other Ranges, Canvases or parts of Canvases in the `items` property. Each entry in the `items` property MUST be a JSON (JavaScript Object Notation) object, and it MUST have the `id` and `type` properties. If a top level Range needs to be dereferenced by the client, then it MUST NOT have the `items` property, such that clients are able to recognize that it should be retrieved in order to be processed.

All of the Canvases or parts that should be considered as being part of a Range MUST be included within the Range's `items` property, or a descendant Range's `items`.

The Canvases and parts of Canvases need not be contiguous or in the same order as in the Manifest's `items` property or any other Range. Examples include newspaper articles that are continued in different sections, a chapter that starts half way through a page, or time segments of a single canvas that represent different sections of a piece of music.

Ranges MAY link to an Annotation Collection that has the content of the Range using the `supplementary` property. The referenced Annotation Collection will contain Annotations that target areas of Canvases within the Range and link content resources to those Canvases.

```
{
  "@context": [
    "http://www.w3.org/ns/anno.jsonld",
    "http://iiif.io/api/presentation/3/context.json"
  ],
  "id": "https://example.org/iiif/book1/manifest",
  "type": "Manifest",
  // Metadata here ...

  "items": [
    // Canvases here ...
  ],

  "structures": [
    {
      "id": "https://example.org/iiif/book1/range/r0",
      "type": "Range",
      "label": { "en": [ "Table of Contents" ] },
      "items": [
        {
          "id": "https://example.org/iiif/book1/canvas/cover",
          "type": "Canvas"
        },
        {
          "id": "https://example.org/iiif/book1/range/r1",
          "type": "Range",
          "label": { "en": [ "Introduction" ] },
          "supplementary": {
            "id": "https://example.org/iiif/book1/annocoll/introTexts",
            "type": "AnnotationCollection"
          },
          "items": [
            {
              "id": "https://example.org/iiif/book1/canvas/p1",
              "type": "Canvas"
            },
            {
              "id": "https://example.org/iiif/book1/canvas/p2",
              "type": "Canvas"
            },
            {
              "type": "SpecificResource",
              "source": "https://example.org/iiif/book1/canvas/p3",
              "selector": {
                "type": "FragmentSelector",
                "value": "xywh=0,0,750,300"
              }
            }
          ]
        },
        {
          "id": "https://example.org/iiif/book1/canvas/backCover",
          "type": "Canvas"
        }
      ]
    }
  ]
}
```

5.5. Annotation Pages

Association of images and other content with their respective Canvases is done via Annotations. Traditionally Annotations are used for associating commentary with the resource the Annotation's text or body is about, the Web Annotation (http://w3.org/TR/annotation-model/) model allows any resource to be associated with any other resource, or parts thereof, and it is reused for both commentary and painting resources on the Canvas. Other resources beyond images might include the full text of the object, musical notations, musical performances, diagram transcriptions, commentary annotations, tags, video, data and more.

These Annotations are collected together in Annotation Page resources, which are included in the `items` property from the Canvas. Each Annotation Page can be embedded in its entirety, if the Annotations should be processed as soon as possible when the user navigates to that Canvas, or a reference to an external page. This reference MUST include `id` and `type`, MUST NOT include `items` and MAY include other properties, such as `behavior`. All of the Annotations in the

Annotation Page SHOULD have the Canvas as their `target`. Embedded Annotation Pages SHOULD be processed by the client first, before externally referenced pages. Clients SHOULD process the Annotation Pages and their items in the order given in the Canvas.

An Annotation Page MUST have an HTTP (Hypertext Transfer Protocol)(S) URI given in `id`, and MAY have any of the other properties defined in this specification or the Web Annotation specification. The Annotations are listed in the `items` property of the Annotation Page.

```
{
  "@context": [
    "http://www.w3.org/ns/anno.jsonld",
    "http://iiif.io/api/presentation/3/context.json"
  ],
  "id": "https://example.org/iiif/book1/annopage/p1",
  "type": "AnnotationPage",

  "items": [
    {
      "id": "https://example.org/iiif/book1/annopage/p1/a1",
      "type": "Annotation"
      // ...
    },
    {
      "id": "https://example.org/iiif/book1/annopage/p1/a2",
      "type": "Annotation"
      // ...
    }
  ]
}
```

5.6. Annotations

Annotations follow the Web Annotation (http://w3.org/TR/annotation-model/) data model. The description provided here is a summary plus any IIIF (International Image Interoperability Framework) specific requirements. It must be noted that the W3C standard is the official documentation.

Note that the Web Annotation data model defines different patterns for the `value` property, when used within an Annotation. The value of a Textual Body or a Fragment Selector, for example, are strings rather than JSON (JavaScript Object Notation) objects with languages and values. Care must be taken to use the correct string form in these cases.

Annotations MUST have their own HTTP (Hypertext Transfer Protocol)(S) URIs, conveyed in the `id` property. The JSON-LD (JSON for Linking Data) description of the Annotation SHOULD be returned if the URI is dereferenced, according to the Web Annotation Protocol (http://w3.org/TR/annotation-protocol/).

The content resource is linked in the `body` of the Annotation. The content resource MUST have an `id` property, with the value being the URI at which it can be obtained. A Canvas MAY be treated as a content resource for the purposes of annotating it on to other Canvases. In this situation, the Canvas MAY be embedded within the Annotation, or require dereferencing to obtain its description.

The type of the content resource MUST be included, and SHOULD be taken from the table listed under the definition of `type`. The format of the resource SHOULD be included and, if so, SHOULD be the media type that is returned when the resource is dereferenced. Content resources MAY also have any of the other properties defined in this specification, including commonly `label`, `summary`, `metadata`, `rights` and `requiredStatement`.

If the content resource is an Image, and a IIIF (International Image Interoperability Framework) Image service is available for it, then the URI MAY be a complete URI to any particular representation made available, such as `https://example.org/image1/full/1000,/0/default.jpg`, but MUST NOT be just the URI of the IIIF (International Image Interoperability Framework) Image service. It MUST have a `type` of “Image”. Its media type MAY be listed in `format`, and its height and width MAY be given as integer values for `height` and `width` respectively. The image then SHOULD have the service referenced from it.

The URI of the Canvas MUST be repeated in the `target` property of the Annotation, or the `source` property of a Specific Resource used in the `target` property.

Additional features of the Web Annotation (http://w3.org/TR/annotation-model/) data model MAY also be used, such as selecting a segment of the Canvas or content resource, or embedding the comment or transcription within the Annotation. The use of these advanced features sometimes results in situations where the `target` is not a content resource, but instead a `SpecificResource`, a `Choice`, or other non-content object. Implementations should check the `type` of the resource and not assume that it is always content to be rendered.

```
{
  "@context": [
    "http://www.w3.org/ns/anno.jsonld",
    "http://iiif.io/api/presentation/3/context.json"
  ],
  "id": "https://example.org/iiif/book1/annotation/p0001-image",
  "type": "Annotation",
  "motivation": "painting",
  "body": {
    "id": "https://example.org/iiif/book1/res/page1.jpg",
    "type": "Image",
    "format": "image/jpeg",
    "service": [
      {
        "id": "https://example.org/images/book1-page1",
        "type": "ImageService3",
        "profile": "level2"
      }
    ],
    "height": 2000,
    "width": 1500
  },
  "target": "https://example.org/iiif/book1/canvas/p1"
}
```

5.7. Annotation Collection

Annotation Collections represent groupings of Annotation Pages that should be managed as a single whole, regardless of which Canvas or resource they target. This allows, for example, all of the Annotations that make up a particular translation of the text of a book to be collected together. A client might then present a user interface that allows all of the Annotations in an Annotation Collection to be displayed or hidden according to the user’s preference.

Note that the Web Annotation Data Model uses the JSON-LD (JSON for Linking Data) 1.0 specification and defines the use of `label` on an Annotation Collection to be one or more strings, rather than a JSON (JavaScript Object Notation) object with languages as keys. Clients SHOULD accept both forms.

Annotation Collections MUST have a URI, and it SHOULD be an HTTP (Hypertext Transfer Protocol)(S) URI. They MUST have a `label` and MAY have any of the other descriptive, linking or rights properties.

```
{
  "@context": [
    "http://www.w3.org/ns/anno.jsonld",
    "http://iiif.io/api/presentation/3/context.json"
  ],
  "id": "https://example.org/iiif/book1/annocoll/transcription",
  "type": "AnnotationCollection",
  "label": [ "Diplomatic Transcription" ],

  "first": { "id": "https://example.org/iiif/book1/annopage/l1", "type": "AnnotationPage" },
  "last": { "id": "https://example.org/iiif/book1/annopage/l120", "type": "AnnotationPage" }
}
```

For Annotation Collections with many Annotations, there will be many pages. The Annotation Collection refers to the first and last page, and then the pages refer to the previous and next pages in the ordered list. Each page is part of the Annotation Collection.

```
{
  "@context": [
    "http://www.w3.org/ns/anno.jsonld",
    "http://iiif.io/api/presentation/3/context.json"
  ],
  "id": "https://example.org/iiif/book1/annopage/l1",
  "type": "AnnotationPage",
  "partOf": {
    "id": "https://example.org/iiif/book1/annocoll/transcription",
    "type": "AnnotationCollection"
  },
  "next": {
    "id": "https://example.org/iiif/book1/annopage/l2",
    "type": "AnnotationPage"
  },
  "items": [
    {
      "id": "https://example.org/iiif/book1/annopage/p1/a1",
      "type": "Annotation"
      // ...
    }
  ]
}
```

6. HTTP (Hypertext Transfer Protocol) Requests and Responses

This section describes the RECOMMENDED request and response interactions for the API (Application Programming Interface). The REST (Representational State Transfer) and simple HATEOAS (Hypermedia as the Engine of Application State) approach is followed where an interaction will retrieve a description of the resource, and additional calls may be made by following links obtained from within the description. All of the requests use the HTTP (Hypertext Transfer Protocol) GET method; creation and update of resources is not covered by this specification.

6.1. URI Recommendations

While any HTTP (Hypertext Transfer Protocol)(S) URI is technically acceptable for any of the resources in the API (Application Programming Interface), there are several best practices for designing the URIs for the resources.

- The URI SHOULD use the HTTPS (Hypertext Transfer Protocol (Secure)) scheme, not HTTP (Hypertext Transfer Protocol).
- The URI SHOULD NOT include query parameters or fragments.
- Once published, they SHOULD be as persistent and unchanging as possible.
- Special characters MUST be encoded.

6.2. Requests

Clients MUST NOT attempt to construct resource URIs by themselves, instead they MUST follow links from within retrieved descriptions or elsewhere.

6.3. Responses

The format for all responses is JSON (JavaScript Object Notation), as described above. The different requirements for which resources MUST provide a response is summarized in Appendix A (/api/presentation/3.0/#a-summary-of-metadata-requirements). While some resources do not require their URI to provide the description, it is good practice if possible.

The HTTP (Hypertext Transfer Protocol) `Content-Type` header of the response SHOULD have the value "application/ld+json" (JSON-LD (JSON for Linking Data)) with the `profile` parameter given as the context document: `http://iiif.io/api/presentation/3/context.json`.

```
Content-Type: application/ld+json;profile="http://iiif.io/api/presentation/3/context.json"
```

If this cannot be generated due to server configuration details, then the content-type MUST instead be `application/json` (regular JSON (JavaScript Object Notation)), without a `profile` parameter.

```
Content-Type: application/json
```

The HTTP (Hypertext Transfer Protocol) server MUST follow the CORS (Cross-Origin Resource Sharing) requirements (https://www.w3.org/TR/cors/) to enable browser-based clients to retrieve the descriptions. If the server receives a request with one of the content types above in the Accept header, it SHOULD respond with that content type following the rules of content negotiation (https://tools.ietf.org/html/rfc7231#section-5.3.2). Recipes for enabling CORS (Cross-Origin Resource Sharing) and conditional Content-Type headers are provided in the Apache HTTP (Hypertext Transfer Protocol) Server Implementation Notes (/api/annex/notes/apache/).

Responses SHOULD be compressed by the server as there are significant performance gains to be made for very repetitive data structures.

7. Authentication

It may be necessary to restrict access to the descriptions made available via the Presentation API (Application Programming Interface). As the primary means of interaction with the descriptions is by web browsers using XmlHttpRequests across domains, there are some considerations regarding the most appropriate methods for authenticating users and authorizing their access. The approach taken is described in the Authentication (/api/auth/) specification, and requires requesting a token to add to the requests to identify the user. This token might also be used for other requests defined by other APIs.

It is possible to include Image API (Application Programming Interface) service descriptions within the Manifest, and within those it is also possible to include links to the Authentication API (Application Programming Interface)'s services that are needed to interact with the image content. The first time an Authentication API (Application Programming Interface) service is included within a Manifest, it MUST be the complete description. Subsequent references SHOULD be just the URI of the service, and clients are expected to look up the details from the full description by matching the URI. Clients MUST anticipate situations where the Authentication service description in the Manifest is out of date: the source of truth is the Image Information document, or other system that references the Authentication API (Application Programming Interface) services.

Appendices

A. Summary of Metadata Requirements

Icon	Meaning
●	Required
◒	Recommended
○	Optional
✕	Not Allowed

Descriptive and Rights Properties

	label	metadata	summary	thumbnail	posterCanvas	requiredStatement	rights	logo
`Collection`	●	◒	◒	◒	○	○	○	○
`Manifest`	●	◒	◒	◒	○	○	○	○
`Canvas`	●	○	○	◒	○	○	○	○
`Annotation`	○	○	○	○	✕	○	○	○
`AnnotationPage`	○	○	○	○	✕	○	○	○
`Range`	●	○	○	○	○	○	○	○
`AnnotationCollection`	●	○	○	○	✕	○	○	○
`Image Content`	○	○	○	○	✕	○	○	○
`Other Content`	○	○	○	○	✕	○	○	○

Technical Properties

	id	type	format	height	width	viewingDirection	behavior	navDate
`Collection`	●	●	✕	✕	✕	✕	○	○
`Manifest`	●	●	✕	✕	✕	○	○	○
`Canvas`	●	●	✕	●	●	✕	○	✕

	id	type	format	height	width	viewingDirection	behavior	navDate
`Annotation`	◓	●	✗	✗	✗	✗	○	✗
`Annotation Page`	●	●	✗	✗	✗	✗	○	✗
`Range`	●	●	✗	✗	✗	○	○	✗
`Annotation Collection`	●	●	✗	✗	✗	○	○	✗
`Image Content`	●	●	○	○	○	✗	○	✗
`Other Content`	●	●	○	○	○	✗	○	✗

Linking Properties

	seeAlso	service	homepage	rendering	partOf	start
`Collection`	○	○	○	○	○	✗
`Manifest`	○	○	○	○	○	✗
`Canvas`	○	○	○	○	○	✗
`Annotation`	○	○	○	○	○	✗
`Annotation Page`	○	○	○	○	○	✗
`Range`	○	○	○	○	○	○
`Annotation Collection`	○	○	○	○	○	✗
`Image Content`	○	○	○	○	○	✗
`Other Content`	○	○	○	○	○	✗

Structural Properties

> Re-write this with items, structures, annotations

Protocol Behavior

	id is dereferenceable
`Collection`	●
`Manifest`	●
`Canvas`	◓
`Annotation`	◓
`Annotation Page`	●
`Range`	○
`Annotation Collection`	○
`Image Content`	●
`Other Content`	●

B. Example Manifest Response

Rebuild the example

C. Versioning

Starting with version 2.0, this specification follows Semantic Versioning (http://semver.org/spec/v2.0.0.html). See the note Versioning of APIs (/api/annex/notes/semver/) for details regarding how this is implemented.

D. Acknowledgements

Many thanks to the members of the IIIF (International Image Interoperability Framework) community (http://iiif.io/community/) for their continuous engagement, innovative ideas and feedback.

E. Change Log

Date	Description
XXXX-YY-ZZ	Version 3.0
2017-06-09	Version 2.1.1 View change log (/api/presentation/3.0/change-log/)
2016-05-12	Version 2.1 (Hinty McHintface) View change log (/api/presentation/2.1/change-log/)
2014-09-11	Version 2.0 (Triumphant Giraffe) View change log (/api/presentation/2.0/change-log/)
2013-08-26	Version 1.0 (unnamed)
2013-06-14	Version 0.9 (unnamed)