W3C Working Group Note

Web Publications

W3C Working Group Note 13 August 2019

This version:

https://www.w3.org/TR/2019/NOTE-wpub-20190813/

Latest published version:

https://www.w3.org/TR/wpub/

Latest editor's draft:

https://w3c.github.io/wpub/

Previous version:

https://www.w3.org/TR/2019/WD-wpub-20190614/

Editors:

Matt Garrish (DAISY Consortium)

Ivan Herman (W3C)

Participate:

GitHub w3c/wpub

File a bug

Commit history

Pull requests

Abstract

The primary objective of this specification is to define requirements for the production of Web Publications. In doing so, it also defines a framework for creating packaged publication formats, such as EPUB and audiobooks, where a pathway to the Web is highly desirable but not necessarily the primary method of interchange or consumption.

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at https://www.w3.org/TR/.

W3C Working Group Note

Due to the lack of practical business cases for Web Publications, and the consequent lack of commitment to implement the technology, the Publishing Working Group has chosen to publish this document as a Note and focus on other areas of interest, including developing the manifest format as a separate specification.

This document was still a work in progress at the time of its publication. As a result, anyone seeking to create Web Publications, or implement a reader for them, should read the approach and proposals outlined in this document with an abundance of caution. It is being published to archive the work and allow incubation, should interest emerge in the future to resume its development.

The following aspects of the specification, in particular, were being actively discussed at the time of publication and are considered incomplete or in need of more review:

- the nature of the canonical identifier;
- whether or not the address should specify a specific resource or only refer to a directory (i.e., rely on a default document being served);
- differences in linking from the primary entry page to the manifest and from a publication resource to the primary entry page; and
- the processing of the manifest, in particular the inheritance of information from the primary entry page (e.g., title or language information).

As well, reviews of the following areas were intended to be undertaken only when the specification stabilized:

- security, such as issues around cross-origin resource sharing and the exact origin for Web Publication resources;
- privacy, both as it relates to the maintenance of stored content and user-identifiable information; and
- user agent implementation of features.

This document was published by the Publishing Working Group as a Working Group Note.

GitHub Issues are preferred for discussion of this specification. Alternatively, you can send comments to our mailing list. Please send them to public-publ-wg@w3.org (archives).

Table of Contents

W3C Working Group Note

PART I: Publication Manifest

1. Introduction §

1.1 Scope §

This specification defines three key concepts:

1. a general manifest format to describe publications;
2. a concrete format using the manifest to represent publications on the Web (Web Publications); and

3. the rules for using the manifest as the basis for modular extensions that desire a pathway to the Web.

This specification does not attempt to constrain the nature of the publications that can be produced—any type of work that can be represented using Web technologies is in scope. It is also designed to be adaptable to the needs of specific areas of publishing, such as audiobook production, and encourages a modular approach for creating specializations.

As much as possible, this specification leverages existing Open Web Platform technologies to achieve its goal—that being to allow for a measure of boundedness — on and off the Web — without changing the way that the Web itself operates.

Moreover, this specification is designed to adapt automatically to updates to Open Web Platform technologies in order to ensure that conforming publications continue to interoperate seamlessly as the Web evolves (e.g., by referencing the latest published versions instead of specific dated versions).

This specification is also intended to facilitate different user agent architectures for the consumption of Web Publications, or any format derived therefrom. While a primary goal is that traditional Web user agents (browsers) will be able to consume Web Publications, this should not limit the capabilities of any other possible type of user agent (e.g., applications, whether standalone or running within a user agent, or even Web Publications that include their own user interface). As a result, the specification does not attempt to architect required solutions for situations whose expected outcome will vary depending on the nature of the user agent and the expectations of the user (e.g., how to prompt to initiate a Web Publication, or at what point or how much of a Web Publication to cache for offline use).

This specification does not define how user agents are expected to render Web Publications. Details about the types of affordances that user agents can provide to enhance the reading experience for users are instead defined in [PWP-UCR].

1.2 Organization §

This section is non-normative.

This specification organized into three distinct parts, each of which builds on the previous:

Part I — Publication Manifest

The first part of this specification defines a general manifest format for expressing information about a digital publication. It uses [schema.org] metadata augmented to include various structural properties about publications, and is serialized in [JSON-LD]. This definition is designed to be the basis for all the specific implementations of digital publications. It allows for interoperability between the formats while accommodating variances in the information that needs to be expressed. The actual manifest requirements for a digital publication format are defined by its respective specification.

Part II — Web Publications

The second part of this specification details the Web Publications format for defining and deploying publications on the Web. It explains the requirements for expressing the Web Publication manifest, as well as various implementation details such as the primary entry page and the table of contents. Web Publications are the Web deployment format for all digital publications based on a Publication Manifest, so all such extensions have to remain compatible with the requirements defined in this section.

Part III — Modular Extensions

The third part of this specification defines how create new packaged digital publication formats using the Publication Manifest model. While such formats are not required to be conforming Web Publications in their native state, it has to be possible to create content that can conform to both requirements.

1.3 Terminology §

This document uses terminology defined by the W3C Note "Publishing and Linking on the Web" [publishing-linking], including, in particular, user, user agent, browser, and address.

Digital Publication

The term digital publication is used to refer to the encoding of a publication in any format that conforms to this specification, whether a Web Publication or a modular extension. All such digital publications share the common manifest format, but differ in their structural and content requirements.

Manifest

A manifest represents structured information about a publication, such as informative metadata, a list of all resources, and a default reading order.

Non-empty

For the purposes of this specification, non-empty is used to refer to an element, attribute or property whose text content or value consists of one or more characters after whitespace normalization, where whitespace normalization rules are defined per the host format.

Web Publication

A Web Publication is a collection of one or more resources, organized together through a manifest into a single logical work with a default reading order. The Web Publication is uniquely identifiable and presentable using Open Web Platform technologies.

1.4 Conformance §

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words *MAY*, *MUST*, *MUST NOT*, *OPTIONAL*, *RECOMMENDED*, *REQUIRED*, *SHOULD*, and *SHOULD NOT* in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. PART I: Publication Manifest §

2.1 Introduction §

This section is non-normative.

A digital publication is described by its manifest, which provides a set of properties expressed using the JSON-LD [json-ld] format (a variant of JSON [ecma-404] for linked data).

The manifest includes both descriptive properties about the publication, such as its title and author, as well as information about the nature and structure of the publication.

This section describes the construction requirements for manifests, and outlines the general set of properties for use with them.

2.2 Authored and Canonical Manifests §

This section is non-normative.

Depending on the state of a digital publication, its manifest exists in one of two forms:

Authored Manifest

> The Authored Publication Manifest is the serialization of the manifest that the author provides with the digital publication (i.e., prior to be digital publication being processed by a user agent). Note that the author does not have to be human; a machine could automatically produce authored manifests for digital publications.

Canonical Manifest

> The Canonical Publication Manifest is a version of the manifest created by user agents when they process the authored manifest and remove all possible ambiguities and incorporate any missing values that can be inferred from another source.

It is possible that an authored manifest is the equivalent of the canonical manifest if there are no ambiguities or missing information, but a canonical manifest only exists after a user agent has inspected the authored manifest as part of the process of obtaining it.

This part of the specification describes the requirements for creating an authored manifest, regardless of the format of the digital publication. Rules for constructing a canonical manifest from the authored manifest are defined for each specific implementation (e.g., the process for Web Publications is described in § 2.9.3 Generating a Canonical Manifest).

2.3 Web IDL §

Although a digital publication's manifest is authored as [json-ld], a user agent processes this information into an internal data structure, which can be in any language, in order to utilize the properties. The exact manner in which this processing occurs, and how the data is used internally, is user agent-dependent and not defined in this specification.

To simplify the understanding of the manifest format for developers, this specification defines an abstract representation of the data structures employed by the manifest using the Web Interface Definition Language (Web IDL) [webidl-1] — the `PublicationManifest` dictionary.

This definition expresses the expected names, datatypes, and possible restrictions for each member of the manifest. Unlike a typical Web IDL definition, however, user agents are not expected to expose the information in the manifest as an API. The Web IDL language is chosen solely to provide an abstraction of the data model.

> NOTE
>
> It is not necessary to understand the Web IDL definition in order to create digital publications. Authoring requirements are defined in the following sections.

2.3.1 The *PublicationManifest* Dictionary §

WebIDL

```
dictionary PublicationManifest {
    required sequence<DOMString>          type;
             sequence<DOMString>          id;
             sequence<DOMString>          accessMode;
             sequence<DOMString>          accessModeSufficient;
             sequence<DOMString>          accessibilityFeature;
             sequence<DOMString>          accessibilityHazard;
             LocalizableString            accessibilitySummary;
             sequence<CreatorInfo>        artist;
             sequence<CreatorInfo>        author;
             sequence<CreatorInfo>        colorist;
             sequence<CreatorInfo>        contributor;
             sequence<CreatorInfo>        creator;
             sequence<CreatorInfo>        editor;
             sequence<CreatorInfo>        illustrator;
             sequence<CreatorInfo>        inker;
             sequence<CreatorInfo>        letterer;
             sequence<CreatorInfo>        penciler;
             sequence<CreatorInfo>        publisher;
             sequence<CreatorInfo>        readBy;
             sequence<CreatorInfo>        translator;
             sequence<DOMString>          url;
             DOMString                    duration;
             DOMString                    inLanguage;
             TextDirection                inDirection;
             DOMString                    dateModified;
             DOMString                    datePublished;
             ProgressionDirection         readingProgression = "ltr";
    required sequence<LocalizableString>  name;
    required sequence<LinkedResource>     readingOrder;
             sequence<LinkedResource>     resources = [];
             sequence<LinkedResource>     links = [];
};


dictionary CreatorInfo {
             sequence<DOMString>          type;
    required sequence<LocalizableString>  name;
             DOMString                    id;
             DOMString                    url;
};

enum TextDirection {
    "ltr",
    "rtl",
    "auto"
};
```

```
dictionary LocalizableString {
    required DOMString                  value;
             DOMString                  language;
};

enum ProgressionDirection {
    "ltr",
    "rtl"
};
```

2.4 Manifest Contexts §

A digital publication's manifest *MUST* start by setting the JSON-LD context [json-ld]. The context has the following two major components:

- the [schema.org] context: `https://schema.org`
- the publication context: `https://www.w3.org/ns/wp-context`

EXAMPLE 1 : The context declaration.

```
{
    "@context" : ["https://schema.org", "https://www.w3.org/ns/wp-context"],
    …
}
```

The publication context document adds features to the properties defined in Schema.org (e.g., the requirement for the creator property to be order preserving).

EDITOR'S NOTE

As part of the continuous contacts with Schema.org the additional features defined in the publication context file could migrate to the core Schema.org vocabulary.

NOTE

Although Schema.org is often referenced using the `http` URI scheme, the vocabulary is being migrated to use the secure `https` scheme as its default. This specification requires the use of `https` when referencing Schema.org in the manifest.

2.5 Publication Types §

A digital publication's manifest *MUST* define its ***Publication Type*** using the **`type`** term [json-ld]. The type *MAY* be mapped onto `CreativeWork` [schema.org].

EXAMPLE 2 : Setting a publication's type to CreativeWork.

```
{
    "@context" : ["https://schema.org", "https://www.w3.org/ns/wp-context"],
    "type"     : "CreativeWork",
    …
}
```

Schema.org also includes a number of more specific subtypes of `CreativeWork`, such as `Article`, `Book`, `TechArticle`, and `Course`. These *MAY* be used instead of, or in addition to, `CreativeWork`.

EXAMPLE 3 : Setting a publication's type to Book.

```
{
    "@context" : ["https://schema.org", "https://www.w3.org/ns/wp-context"],
    "type"     : "Book",
    …
}
```

Each Schema.org type defines a set of properties that are valid for use with it. To ensure that the manifest can be validated and processed by Schema.org aware processors, the manifest *SHOULD* contain only the properties associated with the selected type.

If properties from more than one type are needed, the manifest *MAY* include multiple type declarations.

EXAMPLE 4 : A publication that combines properties from both Book and VisualArtwork.

```
{
    "@context" : ["https://schema.org", "https://www.w3.org/ns/wp-context"],
    "type"     : ["Book", "VisualArtwork"],
    …
}
```

User agents *SHOULD NOT* fail to process manifests that are not valid to their declared Schema.org type(s).

NOTE

Refer to the Schema.org site for the complete list of `CreativeWork` subtypes.

2.6 Properties §

2.6.1 Introduction §

This section is non-normative.

A digital publication's manifest is defined by a set of properties that describe the basic information a user agent requires to process and render the publication. These properties are categorized as followed:

descriptive properties

Descriptive properties describe aspects of a digital publication, such as its title, creator, and language. These properties are primarily drawn from Schema.org and its hosted extensions [schema.org], so they map to one or several Schema.org properties and inherit their syntax and semantics. (The following property categories typically do not have Schema.org equivalents, so are defined specifically for publications.)

resource categorization

Resource categorization properties describe or identify common sets of resources, such as the resource list and default reading order. These properties refer to one or more resources, such as HTML documents, images, script files, and separate metadata files.

NOTE

The categorization of properties exists only to simplify comprehension of their purpose; the groupings have no relevance outside this specification (i.e., the properties are not actually grouped together in the manifest).

NOTE

Each manifest item drawn from schema.org identifies the property it maps to and includes its defining type in parentheses. Properties are often available in many types, however, as a result of the schema.org inheritance model. Refer to each property definition for more detailed information about where it is valid to use.

Schema.org additionally includes a large number of properties that, though relevant for publishing, are not mentioned in this specification — publication authors can use any of these properties. This document defines only the minimal set of manifest items.

EDITOR'S NOTE

There are discussion on whether a best practices document would be created, referring to more schema.org terms. If so, it should be linked from here.

2.6.2 Value Categories §

This section describes the categories of values that can be used with properties of the Publication Manifest.

2.6.2.1 Literals §

Some manifest properties expect a literal text string as their value — one that is not language-dependent, such as a code value or date. These values are expressed as [json] strings.

Literal values are not changed during canonicalization of the manifest, unlike other values which might be, for example, converted to objects.

2.6.2.2 Numbers §

Some manifest properties expect a number as their value. These values are expressed as [json] numbers.

2.6.2.3 Explicit and Implied Objects §

Various manifest properties are expected to be expressed as [json] objects. Although the use of objects is usually recommended, it is also acceptable to use string values that are interpreted as objects depending on the context. The exact mapping of text values to objects is part of the property or object definitions.

2.6.2.3.1 Localizable Strings §

Some manifest properties expect a localizable text string as their value. These values are expressed either as:

- a single string value;
- an anonymous object with a `value` property containing a the property's text and a `language` property that identifies the language of the text.

In the case of single string values, these represent a implied object whose `value` property is the string's text and whose language will be determined from other information in the manifest.

2.6.2.3.2 Entities §

A common case of implied objects in the Publication Manifest properties set is for creators. The entities responsible for the various aspects of creation are expressed as [schema.org] Person and/or Organization objects. To simplify authoring, however, a simple string value can be used for the entity's name. In this case, the entity is assumed to represent a Person.

EXAMPLE 5 : Using a text string instead of a Person object.

The following author name is expressed as a text string:

```
{
    "author" : "Herman Melville",
    …
}
```

but, in the context of creators, it is equivalent to:

```
{
    "author" : {
        "type" : "Person",
        "name" : "Herman Melville"
    },
    …
}
```

(See § 2.6.3.4 Creators for further details.)

2.6.2.3.3 LINKS §

With the exception of the descriptive properties, manifest properties typically link to one or more resources. When a property requires a link value, the link *MUST* be expressed in one of the following two ways:

1. as a string encoding the URL of the resources; or
2. as an instance of a `LinkedResource` object that can be used to express the URL, the media type, and other characteristics of the target resource.

In the case of single string values, these represent an implied `LinkedResource` object whose `url` property is set to that string value.

EXAMPLE 6 : Resource list that includes one link using a relative URL as a string ('datatypes.svg') and two that display the various properties of the a LinkedResource object

```
{
    …
    "resources" : [
        "datatypes.svg",
        {
            "type"            : "LinkedResource",
            "url"             : "test-utf8.csv",
            "encodingFormat"  : "text/csv",
            "name"            : "Test Results",
            "description"     : "CSV file containing the full data set used."
        },
        {
            "type"            : "LinkedResource",
            "url"             : "terminology.html",
            "encodingFormat"  : "text/html",
            "rel"             : "glossary"
        }
    ]
}
```

2.6.2.4 URLs §

URLs are used to identify resources associated with a digital publication. They *MUST* be valid URL strings [url].

Manifest URLs are restricted to only the `http` and `https` schemes [url]. URLs *MUST* dereference to a resource, although user agents are not required to dereference all URLs in the manifest.

In the case of relative-URL strings, these are resolved to absolute-URL strings using a base URL [url].

The base URL for relative-URL strings is determined as follows:

- In the case of an embedded manifest, the base URL is the document base URL [html] of the primary entry page of the publication;
- In the case of a linked manifest, the base URL is URL of the manifest resource.

By consequence, relative-URL strings in embedded manifests are resolved against the URL of the primary entry page *unless* the page declares a base URL (i.e., in a `<base>` element in its header).

NOTE

URLs allow for the usage of characters from Unicode following [rfc3987]. See the note in the HTML5 specification for further details.

2.6.2.5 Identifiers §

Identifiers are URL records [url] that can be used to refer to Web Content in a persistent and unambiguous manner. URLs, URNs, DOIs, ISBNs, and PURLs are all examples of persistent identifiers frequently used in publishing.

2.6.2.6 Arrays §

Some manifest properties allow one or more value of their respective type (literal, object, or URL). As a general rule, these values can be expressed as [json] arrays. When the property value is an array with a single element, however, the array syntax *MAY* be omitted.

> EXAMPLE 7 : Using a text string instead of an array
>
> As a digital publication typically contains many resources, this declaration of a single resource:
>
> ```
> {
>     "resources" : "datatypes.svg",
>     …
> }
> ```
>
> is equivalent to the array:
>
> ```
> {
>     "resources" : ["datatypes.svg"],
>     …
> }
> ```

2.6.3 Descriptive Properties §

2.6.3.1 Accessibility §

The accessibility properties provides information about the suitability of a digital publication for consumption by users with varying preferred reading modalities. These properties typically supplement an evaluation against established accessibility criteria, such as those provided in [WCAG20]. (For linking to a detailed accessibility report, see § 2.7.2.1 Accessibility Report.)

The following properties are categorized as accessibility properties:

Term	Description	Required Value	Value Category	[schema.org] Mapping
accessMode	The human sensory perceptual system or cognitive faculty through which a person may process or perceive information.	One or more text(s).	Array of Literals	accessMode (CreativeWork)
accessModeSufficient	A list of single or combined accessModes that are sufficient to understand all the intellectual content of a resource.	One or more ItemList.	Array of Literals	accessModeSufficient (CreativeWork)
accessibilityFeature	Content features of the resource, such as accessible media, alternatives and supported enhancements for accessibility.	One or more text(s).	Array of Literals	accessibilityFeature (CreativeWork)
accessibilityHazard	A characteristic of the described resource that is physiologically dangerous to some users.	One or more text(s).	Array of Literals	accessibilityHazard (CreativeWork)
accessibilitySummary	A human-readable summary of specific	Text.	Localizable String	accessibilitySummary (CreativeWork)

Term	Description	Required Value	Value Category	[schema.org] Mapping
	accessibility features or deficiencies, consistent with the other accessibility metadata but expressing subtleties such as “short descriptions are present but long descriptions will be needed for non-visual users” or “short descriptions are present and no long descriptions are needed.”			

NOTE

Detailed descriptions of these properties, including the expected values to use with them, are available at [webschemas-a11y].

NOTE

The author can also provide a reference to a detailed Accessibility Report if more information is needed than can be expressed by these properties.

EXAMPLE 8 : Example accessiblity metadata for a document with text and images. The publication provides alternative text and long descriptions appropriate for each image, so can also be read in purely textual form.

```
{
    "@context" : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"     : "CreativeWork",
    …
    "accessMode"            : ["textual", "visual"],
    "accessModeSufficient"  : [
        {
            "type"           : "ItemList",
            "itemListElement": ["textual", "visual"]
        },
        {
            "type"           : "ItemList",
            "itemListElement": ["textual"]
        }
    ],
    …
}
```

2.6.3.2 Address §

A digital publication's ***address*** is a URL that identifies its primary entry page. It is expressed using the `url` property.

Term	Description	Required Value	Value Type	[schema.org] Mapping
`url`	URL of the primary entry page.	A valid URL string [url].	Array of URLs	`url` (Thing)

A digital publication *MAY* have more than one address, but all the addresses *MUST* resolve to the same document.

NOTE
The publication's address can also be used as value for an identifier link relation [link-relation].

EXAMPLE 9 : Setting the address of the main entry page

```
{
    "@context" : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"     : "Book",
    …
    "url"      : "https://publisher.example.org/mobydick",
    …
}
```

2.6.3.3 Canonical Identifier §

A digital publication's ***canonical identifier*** property provides a unique identifier for the publication. It is expressed using the `id` property.

Term	Description	Required Value	Value Type	[schema.org] Mapping
`id`	Preferred version of the publication.	A URL record [url].	Identifier	(None)

The canonical identifier *SHOULD* be a URL that resolves to the preferred version of the digital publication. Using a URL provides a measure of permanence above and beyond a digital publication's address(es). If a digital publication is permanently relocated to a new URL, for example, the canonical address provides a way of discovering the new location (e.g., a DOI registry could be updated with the new URL, or a redirect could be added to the URL of the canonical identifier). It is also intended to provide a means of identifying instances of the same digital publication hosted at different URLs.

> NOTE
>
> Ensuring uniqueness of canonical identifiers is outside the scope of this specification. The actual achievable uniqueness depends on such factors as the conventions of the identifier scheme used and the degree of control over assignment of identifiers.

If a canonical identifier is not provided in the manifest, or the value is an invalid URL, the digital publication does not have a canonical identifier. User agents *MUST NOT* attempt to construct a canonical identifier from any other identifiers provided in the manifest for the canonical manifest.

The specification of the canonical identifier *MAY* be complemented by the inclusion of additional types of identifiers using the `identifier` property [schema.org] and/or its subtypes.

EXAMPLE 10 : Example of setting the canonical identifier and the address as URLs

```
{
    "@context" : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"     : "TechArticle",
    …
    "id"       : "http://www.w3.org/TR/tabular-data-model/",
    "url"      : "http://www.w3.org/TR/2015/REC-tabular-data-model-20151217/",
    …
}
```

EXAMPLE 11 : Example of a URN for the canonical identifier

```
{
    "@context" : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"     : "Book",
    …
    "id"       : "urn:isbn:9780123456789",
    "url"      : "https://publisher.example.org/mobydick",
    …
}
```

2.6.3.4 Creators §

A ***creator*** is an individual or entity responsible for the creation of the digital publication.

The following properties are categorized as creators:

Term	Description	Required Value	Value Category	[schema.org] Mapping
artist	The primary artist for the publication, in a medium other than pencils or digital line art.	One or more `Person`.	Array of Entities	`artist` (VisualArtwork)
author	The author of the publication.	One or more `Person` and/or `Organization`.	Array of Entities	`author` (CreativeWork)
colorist	The individual who adds color to inked drawings.	One or more `Person`.	Array of Entities	`colorist` (VisualArtwork)
contributor	Contributor whose role does not fit to one of the other roles in this table.	One or more `Person` and/or `Organization`.	Array of Entities	`contributor` (CreativeWork)
creator	The creator of the publication.	One or more `Person` and/or `Organization`.	Array of Entities	`creator` (CreativeWork)
editor	The editor of the publication.	One or more `Person`.	Array of Entities	`editor` (CreativeWork)
illustrator	The illustrator of the publication.	One or more `Person`.	Array of Entities	`illustrator` (Book)
inker	The individual who traces over the pencil drawings in ink.	One or more `Person`.	Array of Entities	`inker` (VisualArtwork)

Term	Description	Required Value	Value Category	[schema.org] Mapping
letterer	The individual who adds lettering, including speech balloons and sound effects, to artwork.	One or more `Person`.	Array of Entities	`letterer` (VisualArtwork)
penciler	The individual who draws the primary narrative artwork.	One or more `Person`.	Array of Entities	`penciler` (VisualArtwork)
publisher	The publisher of the publication.	One or more `Person` and/or `Organization`.	Array of Entities	`publisher` (CreativeWork)
readBy	A person who reads (performs) the publication (for audiobooks).	One or more `Person`.	Array of Entities	`readBy` (Audiobook)
translator	The translator of the publication.	One or more `Person` and/or `Organization`.	Array of Entities	`translator` (CreativeWork)

Creators are represented in one of the following two ways:

1. as a string encoding the name of a `Person` [schema.org]; or
2. as an instance of a `Person` or `Organization` [schema.org].

In other words, a single string value is a shorthand for a [schema.org] `Person` whose `name` property is set to that string value. (See also § 2.6.2.3.2 Entities.)

When compiling each set of ***creator information*** from a [schema.org] `Person` or `Organization` type, user agents *MUST* retain the following information when available:

type

One or more strings that identifies the type of creator. This sequence *SHOULD* include "Person" or "Organization".

name

One or more localizable strings for the name of the creator.

id

A canonical identifier of the creator as an Identifier.

url

An address for the creator in the form of a URL.

Note that user agents *MAY* interpret a wider range of creator properties defined by Schema.org than the ones in the preceding list.

The manifest *MAY* include more than one of each type of creator.

EXAMPLE 12 : Author of a book

```
{
    "type"     : "Book",
    "@context" : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    …
    "url"      : "https://publisher.example.org/mobydick",
    "author"   : {
        "type"  : "Person",
        "name"  : "Herman Melville"
    }
}
```

EXAMPLE 13 : Separate listing of editors, authors, and publisher, with some persons expressed as simple strings

```
{
    "@context"   : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"       : "TechArticle",
    …
    "id"         : "http://www.w3.org/TR/tabular-data-model/",
    "url"        : "http://www.w3.org/TR/2015/REC-tabular-data-model-20151217/",
    "author"     : [
        "Jeni Tennison",
        {
            "type"  : "Person",
            "name" : "Gregg Kellogg",
        },{
            "type"  : "Person",
            "name" : "Ivan Herman",
            "id"   : "https://www.w3.org/People/Ivan/"
        }
    ],
    "editor"    : [
        "Jeni Tennison",
        {
            "type"  : "Person",
            "name" : "Gregg Kellogg",
        }
    ],
    "publisher" : {
        "type"  : "Organization",
        "name" : "World Wide Web Consortium",
        "id"   : "https://www.w3.org/"
    }
    …
}
```

2.6.3.5 Duration §

The ***global duration*** indicates the overall length of a *time-based* digital publication (e.g., an audiobook, a book consisting of a series of video clips, etc.). It is expressed as a "Duration" value as defined by [iso8601].

Term	Description	Required Value	Value Category	[schema.org] Mapping
duration	Overall duration of a time-based publication.	Duration value as defined by [iso8601]	Literal	duration (Property)

EXAMPLE 14 : Global duration provided in the manifest (in seconds)

```
{
    "@context"       : ["https://schema.org", "https://www.w3.org/ns/wp-context"],
    "type"           : "Audiobook",
    "id"             : "https://example.org/flatland-a-romance-of-many-dimensions/",
    "url"            : "https://w3c.github.io/wpub/experiments/audiobook/",
    "name"           : "Flatland: A Romance of Many Dimensions",
    …
    "duration"       : "PT15153S",
    …
}
```

NOTE

The relevant Wikiepedia page gives a concise description of the ISO duration syntax.

2.6.3.6 Language and Base Direction §

A digital publication has at least one natural ***language***, which is the language that the content is expressed in (e.g., English, French, Chinese). It also has a natural ***base direction*** in which it is written — the display direction, either left-to-right or right-to-left.

The digital publication manifest includes entries to set both these concepts, which can influence, for example, the behavior of a user agent (e.g., it might place a pop-up for a table of contents on the right hand side for publications whose natural base direction is right-to-left).

NOTE

It is important to differentiate the language *of the publication* from the language and the base direction of the individual resources that compose it. If such resources are, for example, in HTML, the language and direction need to be set in those resources, too. The language and base direction of the publication are not inherited.

Similarly, each natural language property value in the manifest (e.g., title, creators) is a localizable string.

> NOTE
>
> For more information about localized strings on the Web, refer to [string-meta].

The natural language and base direction can be set for both the publication and the natural language properties values of the manifest.

If a user agent requires the language and one is not available in the authored manifest (either globally or specifically for that property), or the obtained value is invalid, the user agent *MAY* attempt to determine the language when generating the canonical manifest. This specification does not mandate how such a language tag is created. The user agent might:

- use the non-empty language declaration of the manifest;
- use the first non-empty language declaration found in a resource in the default reading order;
- calculate the language using its own algorithm.

No default values are specified for the language or the default base direction.

2.6.3.6.1 Global Language and Direction §

The manifest *MAY* include global language and base direction declarations for the publication using the following properties.

Term	Description	Required Value	Value Category	[schema.org] Mapping
inLanguage	Default language for the publication as well as the textual manifest values	language code as defined in [bcp47]	Literal	`inLanguage` (Property)
inDirection	Default base direction for the publication as well as the textual manifest values	`ltr`, `rtl`, or `auto`	Literal	(None)

The natural language *MUST* be a tag that conforms to [bcp47], while the ***base language direction*** *MUST* have one of the following values:

- ***ltr***: indicates that the textual values are explicitly directionally set to left-to-right text;
- ***rtl***: indicates that the textual values are explicitly directionally set to right-to-left text;
- ***auto*** indicates that the textual values are explicitly directionally set to the direction of the first character with a strong directionality, following the rules of the Unicode Bidirectional Algorithm [bidi].

When specified, these properties are also used as defaults for textual values in the manifest.

The global language information *MAY* be overridden by individual values.

If the manifest is embedded in the primary entry page via a `script` element, and the manifest does not set the global language and/or the base direction (see § 2.6.3.6.1 Global Language and Direction), the `lang` and the `dir` attributes of the `script` element are used as the global language and base direction, respectively (see the details on handling the `lang` and `dir` attributes in [html]).

ISSUE 438 : Inheriting (or not) the language tag of a <script> element

(This issue has been noted in the WPUB spec for a while, and was never recorded.)

The current editors' draft says:

> If the manifest is embedded in the primary entry page via a script element, and the manifest does not set the global language and/or the base direction (see § 2.6.3.4.1 Global Language and Direction), the lang and the dir attributes of the script element are used as the global language and base direction, respectively.

It must be noted that the JSON-LD 1.1 draft does not have this behavior, and the `lang` and `dir` attributes of the `<script>` element are ignored. We may want to remove this behavior from WPUB as well, to stay in sync.

NOTE

If authors intend to use a manifest, or a manifest template, both as embedded manifest and as a separate resource, they are strongly encouraged to set these properties explicitly to avoid interference of the containing `script` element in case of embedding.

2.6.3.6.2 Item-specific Language §

It is possible to set the language for any textual value in the manifest. This information *MUST* be set as a ***`localizable string`***, i.e., using the `value` and `language` terms (instead of a simple string) [json-ld]:

EXAMPLE 15 : Setting the author name to French using a localizable string

```
{
    "@context" : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"     : "Book",
    …
    "author" : {
        "type"  : "Person",
        "name" : {
            "value"    : "Marcel Proust",
            "language" : "fr"
        }
    }
}
```

The value of the ***language*** *MUST* be set to a language code as defined in [bcp47].

When used in a context of localizable texts, a simple string value is a shorthand for a `localizable string`, with the ***value*** set to the string value, and the language set to the value of the `inLanguage` property, if applicable, and unset otherwise. In other words, the previous example is equivalent to:

EXAMPLE 16 : Setting the default language of an author name to French

```
{
    "@context"   : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"       : "Book",
    "inLanguage" : "fr",
    …
        "author"     : "Marcel Proust",
    …
}
```

(See also § 2.6.2.3 Explicit and Implied Objects.)

It is *not* possible to set the direction explicitly for a value.

NOTE

Setting the direction for a natural text value is currently not possible in JSON-LD [json-ld]. In case the JSON-LD community, as well as the schema.org community, introduces such a feature, future versions of this specification may extend the ability of manifests to include this.

In order to correctly handle manifests entries containing right-to-left or bidirectional text, user agents *SHOULD* identify the base direction of any given natural language value by scanning the text for the first strong directional character.

> NOTE
>
> In situations where the first-strong heuristics will produce the wrong result (e.g., a string in the Arabic or Hebrew script that begins with a Latin acronym), content developers may want to prepend a Unicode formatting character to the string. This would then produce the necessary base direction when the heuristics are applied. They should use one of the following formatting characters: U+200E LEFT-TO-RIGHT MARK, or U+200F RIGHT-TO-LEFT MARK. (See § D. Examples for bidirectional texts.)

Once the base direction has been identified, user agents *MUST* determine the appropriate rendering and display of natural language values according to the Unicode Bidirectional Algorithm [bidi]. This could require wrapping additional markup or Unicode formatting characters around the string prior to display, in order to apply the base direction.

Once the base direction has been identified, user agents *MUST* determine the appropriate rendering and display of natural language values according to the Unicode Bidirectional Algorithm[bidi]. This could require wrapping additional markup or control characters around the string prior to display, in order to apply the base direction. (See § D. Examples for bidirectional texts.

2.6.3.7 Last Modification Date §

The ***last modification date*** is the date when the digital publication was last updated (i.e., whenever changes were last made to any of the resources of the publication, including the manifest). It is expressed using the `dateModified` property.

Term	Description	Required Value	Value Category	[schema.org] Mapping
`dateModified`	Last modification date of the publication.	A `Date` or `DateTime` value [schema.org], both expressed in ISO 8601 Date, or Date Time formats, respectively [iso8601].	Literal	`dateModified` (CreativeWork)

The last modification date does not necessarily reflect all changes to the publication (e.g., third-party content could change without the author being aware). User agents *SHOULD* check the last modification date of individual resources to determine if they have changed and need updating.

EXAMPLE 17 : Last modification date of the publication

```
{
    "@context"     : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"         : "TechArticle",
    …
    "id"           : "http://www.w3.org/TR/tabular-data-model/",
    "url"          : "http://www.w3.org/TR/2015/REC-tabular-data-model-20151217/",
    "dateModified" : "2015-12-17",
    …
}
```

2.6.3.8 Publication Date §

The ***publication date*** is the date on which the digital publication was originally published. It represents a static event in the lifecycle of a publication and allows subsequent revisions to be identified and compared. It is expressed using the `datePublished` property.

Term	Description	Required Value	Value Category	[schema.org] Mapping
datePublished	Creation date of the publication.	A `Date` or `DateTime`, both expressed in ISO 8601 Date, or Date Time formats, respectively [iso8601].	Literal	`datePublished` (CreativeWork)

The exact moment of publication is intentionally left open to interpretation: it could be when the publication is first made available online or could be a point in time before publication when the publication is considered final.

EXAMPLE 18 : Creation and modification date of the publication

```
{
    "@context"      : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"          : "TechArticle",
    …
    "id"            : "http://www.w3.org/TR/tabular-data-model/",
    "url"           : "http://www.w3.org/TR/2015/REC-tabular-data-model-20151217/",
    "datePublished" : "2015-12-17",
    "dateModified"  : "2016-01-30",
    …
}
```

2.6.3.9 Reading Progression Direction §

The ***reading progression*** establishes the reading direction from one resource to the next within a digital publication. It is expressed using the `readingDirection` property.

Term	Description	Required Value	Value Category	[schema.org] Mapping
readingProgression	Reading direction from one resource to the other.	`ltr` or `rtl`	Literal	(None)

The value of this property *MUST* be either:

- ***ltr***: left-to-right;
- ***rtl***: right-to-left.

The default value is `ltr`.

This property has *no effect* on the rendering of the individual primary resources; it is only relevant for the progression direction from one resource to the other.

> NOTE
>
> The reading progression of a publication is used to adapt such publication level interactions as menu position, swap direction, defining tap zones to lead the user to the next and previous pages, touch gestures, etc.

If the `readingProgression` is not set, user agents *MUST* use the default value `ltr` when generating the canonical manifest.

EXAMPLE 19 : Reading progression set explicitl to ltr

```
{
    "@context"           : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"               : "Book",
    …
    "url"                : "https://publisher.example.org/mobydick",
    "readingProgression" : "ltr"
}
```

2.6.3.10 Title §

The title provides the human-readable name of the digital publication. It is expressed using the `name` property.

Term	Description	Required Value	Value Category	[schema.org] Mapping
name	Human-readable title of the publication.	One or more text items for the title.	Array of Localizable Strings	`name` (Thing)

If a title is not included in the authored manifest, and a digital publication does not define alternative rules for obtaining one, the user agent *MUST* create one. This specification does not specify what heuristics to use to generate such a title.

NOTE

A user agent is not expected to produce a meaningful title [wcag20] for a publication when one is not specified.

EXAMPLE 20 : Title of the book set explicitly

```
{
    "@context" : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"     : "Book",
    …
    "url"      : "https://publisher.example.org/mobydick",
    "name"     : "Moby Dick"
}
```

2.6.4 Resource Categorization Properties §

Publication resources are specified via the default reading order, the resource list, and the `links`, as defined in this section. These lists contain references to informative resources like the privacy policy, and structural resources like the table of contents.

Note that a particular resource's URL *MUST NOT* appear in more than one of these lists, and a URL *MUST NOT* be repeated within a list.

The manifest *MUST NOT* include a reference to itself within any of these lists.

2.6.4.1 Default Reading Order §

The ***default reading order*** is a specific progression through a set of digital publication resources. A user might follow alternative pathways through the content, but in the absence of such interaction the default reading order defines the expected progression from one resource to the next.

The default reading order is expressed using the `readingOrder` property.

Term	Description	Required Value	Value Category	[schema.org] Mapping
		One or more of: • a string, representing the URL of the resource; or • an instance of a `LinkedResource` object		
`readingOrder`		The order of items is *significant*. The URLs *MUST NOT* include fragment identifiers. Non-HTML resources *SHOULD* be expressed as `LinkedResource` objects with their `encodingFormat` values set.	Array of Links	(None)

The default reading order *MUST* include at least one resource.

EXAMPLE 21 : Reading order expressed as a simple list of URLs

```
{
    "@context" : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"     : "Book",
    …
    "url"      : "https://publisher.example.org/mobydick",
    "name"     : "Moby Dick",
    "readingOrder" : [
        "html/title.html",
        "html/copyright.html",
        "html/introduction.html",
        "html/epigraph.html",
        "html/c001.html",
        …
    ]
}
```

EXAMPLE 22 : Reading order expressed as objects providing more information on items

```
{
    "@context" : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"     : "Book",
    …
    "url"      : "https://publisher.example.org/mobydick",
    "name"     : "Moby Dick",
    "readingOrder" : [{
        "type"           : "LinkedResource",
        "url"            : "html/title.html",
        "encodingFormat" : "text/html",
        "name"           : "Title page"
    },{
        "type"           : "LinkedResource",
        "url"            : "html/copyright.html",
        "encodingFormat" : "text/html",
        "name"           : "Copyright page"
    },{
        …
    }]
}
```

2.6.4.2 Resource List §

The ***resource list*** enumerates any additional resources used in the processing and rendering of a digital publication that are not already listed in the default reading order. It is expressed using the `resources` property.

Term	Description	Required Value	Value Category	[schema.org] Mapping
`resources`		One or more of: • a string, representing the URL of the resource; or • an instance of a `LinkedResource` object The order of items is *not significant*. The URLs *MUST NOT* include fragment identifiers. It is *RECOMMENDED* to use `LinkedResource` objects with their `encodingFormat` values set.	Array of Links	(None)

The completeness of the resource list can affect the usability of a digital publication in certain reading scenarios (e.g., the ability to read it offline). For this reason, it is strongly *RECOMMENDED* to provide a comprehensive list of all of the publication's constituent resources beyond those listed in the default reading order.

In some cases, a comprehensive list of these resources might not be easily achieved (e.g., third-party scripts that reference resources from deep within their source), but a user agent *SHOULD* still be able to render a publication even if some of these resources are not identified as belonging to the publication (e.g., if it is taken offline without them).

EXAMPLE 23 : Listing resources, some via a simple URL, some with more details

```
{
    "@context"   : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"       : "TechArticle",
    …
    "id"         : "http://www.w3.org/TR/tabular-data-model/",
    "url"        : "http://www.w3.org/TR/2015/REC-tabular-data-model-20151217/",
    …
    "resources"  : [
        "datatypes.html",
        "datatypes.svg",
        "datatypes.png",
        "diff.html",
        {
            "type"              : "LinkedResource",
            "url"               : "test-utf8.csv",
            "encodingFormat"    : "text/csv"
        },{
            "type"              : "LinkedResource",
            "url"               : "test-utf8-bom.csv",
            "encodingFormat"    : "text/csv"
        },{
            …
        }
    ],
    …
}
```

2.6.4.3 Links §

The ***links*** property provides a list of resources that are *not* required for the processing and rendering of a digital publication (i.e., the content of the publication remains unaffected even if these resources are not available).

Term	Description	Required Value	Value Category	[schema.org] Mapping
links		One or more of:	Array of Links	(None)

Term	Description	Required Value	Value Category	[schema.org] Mapping
		• a string, representing the URL of the resource; or • an instance of a `LinkedResource` object The order of items is *not significant*. It is *RECOMMENDED* to use `LinkedResource` objects with their `encodingFormat` and `rel` values set.		

Linked resources are typically made available to user agents to augment or enhance the processing or rendering, such as:

- a privacy policy or license that the user agent can offer a link to from a shelf;
- a metadata record that the user agent can use to discover and display more information about the publication;
- a dictionary of terms the user agent can process to provide enhanced language help;

Links can also be used to identify resources used in the online rendering of a publication, but that are not essential to include when the publication is taken offline or packaged (e.g., to minimize the size). These include:

- large font files that enhance the appearance of the publication but are not vital to its display (i.e., a fallback font will suffice);
- third-party scripts that are not intended for use when a publication is taken offline or packaged (e.g., tracking scripts).

The `links` list *SHOULD* include resources necessary to render a linked resource (e.g., scripts, images, style sheets).

Resources listed in the `links` list *MUST NOT* be listed in the default reading order or resource list.

User agents *MAY* ignore linked resources, and are not required to take them offline with a publication. These resources *SHOULD NOT* be included when packaging a publication.

2.6.5 Extensibility §

The manifest is designed to provide a basic set of properties for use by user agents in presenting and rendering a digital publication, but *MAY* be extended in the following ways:

1. by the provision of linked metadata records.
2. through the inclusion of additional properties in the manifest;

Although both methods are valid, the use of linked records is *RECOMMENDED*.

This specification does not define how such additional properties are compiled, stored or exposed by user agents in their internal representation of the manifest. A user agent *MAY* ignore some or all extended properties.

2.6.5.1 Linked records §

Extending the manifest through links to a record, such as an ONIX [onix] or BibTeX [bibtex] file, *MUST* be expressed using a `LinkedResource` object, where:

- the `rel` value of the `LinkedResource` *SHOULD* include a relevant identifier defined by IANA or by other organizations; if the link record contains descriptive metadata it *MUST* include the `describedby` (IANA) identifier;
- the value of the `encodingFormat` in the link *MUST* use the MIME media type [rfc2046] defined for that particular type of record, if applicable.

Linked records *MUST* be included in the resource list when they are part of the publication (i.e., are needed for more than just manifest extensibility). Otherwise, they *MUST* be included in the links list.

EXAMPLE 24 : Link to external ONIX for Books Metadata file

```
{
"@context"    : ["https://schema.org","https://www.w3.org/ns/wp-context"],
"type"        : "Book",
…
"url"         : "https://publisher.example.org/mobydick",
"name"        : "Moby Dick",
"links"   : [{
        "type"            : "LinkedResource",
        "url"             : "https://www.publisher.example.org/mobydick-onix.xml",
        "encodingFormat"  : "application/onix+xml",
        "rel"             : "describedby"
},{
        …
}],
…
}
```

EDITOR'S NOTE

The `application/onix+xml` MIME type has not yet been registered by IANA at the time of writing this document, and is included in the example for illustrative purposes only.

2.6.5.2 Additional Properties in the Manifest §

Additional properties can be included directly in the manifest. It is *RECOMMENDED* that these properties be taken from public schemes like [schema.org] or [dcterms] and use values from controlled vocabularies whenever possible. Proprietary terms *MAY* be used, but it is *RECOMMENDED* that such terms be included using Compact IRIs [json-ld], with prefixes defined as part of the context.

EXAMPLE 25 : Usage of the schema.org 'copyrightYear' and 'copyrightHolder' terms, as an extension to the basic data

```
{
"@context"        : ["https://schema.org","https://www.w3.org/ns/wp-context"],
"type"            : "TechArticle",
…
"id"              : "http://www.w3.org/TR/tabular-data-model/",
"url"             : "http://www.w3.org/TR/2015/REC-tabular-data-model-20151217/",
"copyrightYear"   : "2015",
"copyrightHolder" : "World Wide Web Consortium",
…
}
```

EXAMPLE 26 : Usage of the Dublin Core 'subject' with the 2012 ACM Classification terms, as an extension to the basic data

```
{
"@context"    : ["https://schema.org","https://www.w3.org/ns/wp-context"],
"type"        : "CreativeWork",
…
"id"          : "http://www.w3.org/TR/tabular-data-model/",
"url"         : "http://www.w3.org/TR/2015/REC-tabular-data-model-20151217/",
"dc:subject" : ["Web data description languages","Data integration","Data Exchange"],
…
}
```

NOTE

A prefix definition `dc` for [dcterms] is included in the context file of [schema.org]. This means that it is not necessary to add the prefix explicitly. The same is true for a number of other public vocabularies; see the schema.org context file for further details.

2.7 Resource Relations §

2.7.1 Introduction §

This section is non-normative.

The manifest identifies key resources of a digital publication through the use of link relations. These relations are applied to the `rel` property of `LinkedResource` objects (e.g., the links found in the table of contents and resource list).

The types of resources these relations identify are categorized as follows:

informative resources

Informative resources are resources that contain additional information about the publication, such as its privacy policy, accessibility report, or preview.

structural resources

Structural resources are key meta structures of the publication, such as the cover image, table of contents, and page list.

2.7.2 Informative Resources §

2.7.2.1 Accessibility Report §

An accessibility report provides information about the suitability of a digital publication for consumption by users with varying preferred reading modalities. These reports typically identify the result of an evaluation against established accessibility criteria, such as those provided in [WCAG21], and are an important source of information in determining the usability of a publication.

An accessibility report is identified using the `accessibility-report` link relation.

> EDITOR'S NOTE
>
> The `accessibility-report` term is not currently registered in the IANA link relations but the Working Group expects to add it.

The manifest *SHOULD* include a link to an accessibility report when one is available for a publication. It is *RECOMMENDED* that the report be included as a resource of the publication.

It is also *RECOMMENDED* that the accessibility report be provided in a human-readable format, such as [html]. Augmenting these reports with machine-processable metadata, such as provided in Schema.org [schema.org], is also *RECOMMENDED*.

EXAMPLE 27 : Link to an accessibility report

```
{
    "@context"   : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"       : "Book",
    …
    "url"        : "https://publisher.example.org/mobydick",
    "name"       : "Moby Dick",
    "links"  : [{
        "type"        : "LinkedResource",
        "url"         : "https://www.publisher.example.org/mobydick-accessibility.html",
        "rel"         : "accessibility-report"
    },{
        …
    }],
    …
}
```

2.7.2.2 Preview §

Not all digital publications will be available to all users (e.g., they might be restricted to registered users of a site). In such cases, the publisher might wish to provide a preview of the content in order to entice users to access the full version.

A ***preview*** is identified using the `preview` link relation [iana-link-relations].

Previews *MAY* be located externally or included as resources of digital publications.

EXAMPLE 28 : A preview is identified as an audio resource of a digital publication

```
"@context"   : ["https://schema.org","https://www.w3.org/ns/wp-context"],
"type"       : "Book",
…
"url"        : "https://publisher.example.org/mobydick",
"name"       : "Moby Dick",
"links"  : [{
        "type"            : "LinkedResource",
        "url"             : "preview.mp3",
        "encodingFormat"  : "audio/mpeg",
        "rel"             : "preview"
},{
        …
}],
…
}
```

EXAMPLE 29 : A preview is expressed as an external link

```
"@context"   : ["https://schema.org","https://www.w3.org/ns/wp-context"],
"type"       : "Book",
…
"url"        : "https://publisher.example.org/mobydick",
"name"       : "Moby Dick",
"links"  : [{
        "type"            : "LinkedResource",
        "url"             : "https://publisher.example.org/mobydickpreview.html",
        "encodingFormat"  : "text/html",
        "rel"             : "preview"
},{
        …
}],
…
}
```

2.7.2.3 Privacy Policy §

Users often have the legal right to know and control what information is collected about them, how such information is stored and for how long, whether it is personally identifiable, and how it can be expunged. Including a statement that addresses such privacy concerns is consequently an important part of publishing digital publications. Even if no information is collected, such a declaration increases the trust users have in the content.

A link to a privacy policy can be included in the manifest for this purposes. It is *RECOMMENDED* that the privacy policy be included as a resource of the publication.

A ***privacy policy*** is identified using the `privacy-policy` link relation [iana-link-relations].

Refer to § 3.8 Privacy for more information about privacy considerations in publications.

EXAMPLE 30 : Privacy policy expressed as an external link

```
{
    "@context"   : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"       : "TechArticle",
    …
    "id"         : "http://www.w3.org/TR/tabular-data-model/",
    "url"        : "http://www.w3.org/TR/2015/REC-tabular-data-model-20151217/",
    …
    "links"  : [{
        "type"           : "LinkedResource",
        "url"            : "https://www.w3.org/Consortium/Legal/privacy-statement-20140324",
        "encodingFormat" : "text/html",
        "rel"            : "privacy-policy"
    },{
            …
    }],
    …
}
```

2.7.3 Structural Resources §

2.7.3.1 Cover §

The ***cover*** is a resource that user agents can use to present the digital publication (e.g., in a library or bookshelf, or when initially loading the publication).

The cover is identified by the `cover` link relation. The URL expressed in the `url` term *MUST NOT* include a fragment identifier.

> EDITOR'S NOTE
>
> The `cover` term is not currently registered in the IANA link relations but the Working Group expects to add it.

If the cover is in an image format, a `title` and `description` *SHOULD* be provided. User agents can use these properties to provide alternative text and descriptions when necessary for accessibility.

More than one cover *MAY* be referenced from the manifest (e.g., to provide alternative formats and sizes for different device screens). If multiple covers are specified, each instance *MUST* define at least one unique property to allow user agents to determine its usability (e.g., a different format, height, width or relation).

EXAMPLE 31 : Cover HTML page

```
{
    "@context"   : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"       : "Book",
    …
    "url"        : "https://publisher.example.org/donquixote",
    "name"       : "Don Quixote",
    "resources"  : [{
        "type"           : "LinkedResource",
        "url"            : "cover.html",
        "encodingFormat" : "text/html",
        "rel"            : "cover"
    },{
        …
    }],
    …
}
```

EXAMPLE 32 : Cover image with title and description

```
{
    "@context"   : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"       : "Book",
    …
    "url"        : "https://publisher.example.org/mobydick",
    "name"       : "Moby Dick",
    "resources"  : [{
        "type"           : "LinkedResource",
        "url"            : "whale-image.jpg",
        "encodingFormat" : "image/jpeg",
        "rel"            : "cover",
        "name"           : "Moby Dick attacking hunters",
        "description"    : "A white whale is seen surfacing from the water to attack a small wh
    },{
        …
    }],
    …
}
```

EXAMPLE 33 : Cover image in JPEG and SVG formats

```
{
    "@context"   : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"       : "Book",
    …
    "url"        : "https://publisher.example.org/donquixote",
    "name"       : "Gulliver's Travels",
    "resources"  : [{
        "type"           : "LinkedResource",
        "url"            : "lilliput.jpg",
        "encodingFormat" : "image/jpeg",
        "rel"            : "cover"
    },{
        "type"           : "LinkedResource",
        "url"            : "lilliput.svg",
        "encodingFormat" : "image/svg+xml",
        "rel"            : "cover"
    },{
        …
    }],
    …
}
```

2.7.3.2 Page List §

The page list is a navigational aid that contains a list of static page demarcation points within a digital publication.

The page list is identified by the `pagelist` link relation. The URL expressed in the `url` term *MUST NOT* include a fragment identifier.

> EDITOR'S NOTE
>
> The `pagelist` term is not currently registered in the IANA link relations but the Working Group expects to add it.

The link to the page list *MAY* be specified in either the default reading order or resource-list, but *MUST NOT* be specified in both.

EXAMPLE 34 : Page list identified in another resource of the publication

```
{
"@context"   : ["https://schema.org","https://www.w3.org/ns/wp-context"],
"type"       : "Book",
…
"url"        : "https://publisher.example.org/mobydick",
"name"       : "Moby Dick",
"resources"  : [{
        "type"       : "LinkedResource",
        "url"        : "toc_file.html",
        "rel"        : "pagelist"
},{
        …
}],
…
}
```

2.7.3.3 Table of Contents §

The table of contents is a navigational aid that provides links to the majort structural sections of a digital publication.

The ***table of contents*** is identified by the `contents` link relation [iana-link-relations]. The URL expressed in the `url` term *MUST NOT* include a fragment identifier.

The link to the table of contents *MAY* be specified in either the default reading order or resource-list, but *MUST NOT* be specified in both.

EXAMPLE 35 : Resource containing the table of contents identified by its rel attribute value

```
{
    "@context"   : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"       : "Book",
    …
    "url"        : "https://publisher.example.org/mobydick",
    "name"       : "Moby Dick",
    "resources"  : [{
        "type"       : "LinkedResource",
        "url"        : "toc_file.html",
        "rel"        : "contents"
    },{
        …
    }],
    …
}
```

2.7.4 Extensions §

If additional relations beyond those defined in this specification need to be expressed, the `rel` property can be extended in one of the following ways:

- through the use of relations defined in [iana-relations]; or
- through the use of extension relation types [rfc8288].

Use of relations from [iana-relations] is *RECOMMENDED*.

2.8 Association §

2.8.1 Linking §

Links to a manifest *MUST* take one or both of the following forms:

- An HTTP `Link` header field [rfc5988] with its `rel` parameter set to the value "`publication`".

  EXAMPLE 36

  ```
  Link: <https://example.com/webpub/manifest>; rel=publication
  ```

- A `link` element [html] with its `rel` attribute set to the value "`publication`".

  EXAMPLE 37

  ```
  <link href="https://example.com/webpub/manifest" rel="publication"/>
  ```

When a manifest is embedded within an HTML document, the link *MUST* include a fragment identifier that references the `script` element that contains the manifest (see § 2.8.2 Embedding).

EXAMPLE 38 : Link to a manifest within the same HTML resource

```
<link href="#example_manifest" rel="publication">
…
<script id="example_manifest" type="application/ld+json">
{
       "@context" : ["https://schema.org", "https://www.w3.org/ns/wp-context"],
       …
}
</script>
```

ISSUE 132 : Using rel="publication" topic:manifest

The exact value of `rel` is still to be agreed upon and should be registered by IANA.

EDITOR'S NOTE

The following details might be moved to the lifecycle section in a future draft.

When a resource links to multiple manifests, a user agent *MAY* choose to present one or more alternatives to the end user, or choose a single alternative on its own. The user agent *MAY* choose to present any manifest based upon information that it possesses, even one that is not explicitly listed as a parent (e.g., based upon information it calculates or acquires out of band). In the absence of a preference by user agent implementers, selection of the first manifest listed is suggested as a default.

2.8.2 Embedding §

When a manifest is embedded within an HTML document, it *MUST* be included in a `script` element [html] whose `type` attribute is set to `application/ld+json`.

Additionally, the `script` element *MUST* include a unique identifier in an `id` attribute [html]. This identifier ensures that the manifest can be referenced.

EXAMPLE 39 : A Web Publication Manifest included in an HTML document

```
<script id="example_manifest" type="application/ld+json">
   {
      …
   }
</script>
```

2.9 Publication Manifest Lifecycle §

2.9.1 Introduction §

This section is non-normative.

This section describes the steps a user agent follows to process an authored manifest into an internal representation of the data structure it contains.

The first step in this process is to obtain the manifest, the exact steps by which to do so are typically defined by each digital publication format.

The process then involves generating a canonical form of the manifest, which is a representation that adds any missing data not explicitly authored (e.g., information could be gleaned from a containing HTML document if the manifest is embedded inside a `script` tag).

After a canonical manifest is generated, the data is put through a final set of post-processing steps to check its validity, ultimately resulting in a data structure that the user agent can use.

Within this process are various extension points that allow digital publication formats to enhance the basic requirements for their own specialized needs and audiences.

2.9.2 Processing a Manifest §

The ***steps for processing a manifest*** are given by the following algorithm. The algorithm, if successful, returns a processed manifest; otherwise, it terminates prematurely and returns nothing. In the case of nothing being returned, the user agent *MUST* ignore the manifest declaration.

The algorithm takes the following arguments:

- *text*: a UTF-8 string containing the manifest;
- *base*: a URL string that represents the base URL for the manifest.
- *document*: the HTML Document (DOM) Node of the document that references the manifest.

1. Let *json* be the result of parsing *text*. If parsing throws an error, terminate this algorithm.
2. If Type(*json*) is not `Object`, terminate this algorithm.
3. Let *canonical manifest* be the canonical manifest derived from *json*, using the values of *json*, *base*, and `document` as input to the algorithm described in § 2.9.3 Generating a Canonical Manifest.
4. Check whether the *canonical manifest* fulfills the minimal requirements for a Publication Manifest, namely:
    - the JSON-LD context is set as required in § 2.4 Manifest Contexts;
    - the Publication type is set as required in § 2.5 Publication Types; and
    - any extension requirements are set as defined in their respective specifications.

    If any of these requirements is not met, terminate the algorithm.
5. Let *processed manifest* be the result of post-processing a canonical manifest given *canonical manifest*.
6. Return *processed manifest*.

> NOTE
>
> The algorithm does not describes how error and warning messages should be reported. This is implementation dependent.

2.9.3 Generating a Canonical Manifest §

The steps to convert a Publication Manifest into a Canonical Manifest are given by the following algorithm. The algorithm takes the following arguments:

- *manifest*: a JSON object that represent the manifest
- *base*: a URL string that represents the base URL for the manifest
- *document*: the HTML Document (DOM) Node of the document that references the manifest.

The steps of the algorithm are described below. The algorithm varies from strict JavaScript notation in that *P["term"]* refers to the value in the object *P* for the label *"term"*, where *P* is either *manifest* or an object appearing within *manifest* (e.g., a *Person*). The algorithm replaces or adds some terms to *manifest*; the replacement terms are expressed in JSON syntax as `{"term":"value"}`.

1. let *lang* string represent the default language, set to:
    - the value of the `lang` value for the `script` element in `document` (when set); or
    - `undefined` otherwise

    > **Explanation**
    >
    > This value is used in the step on language below.

2. let *dir* string represents the base direction, set to:
    - the value of the `dir` value for the `script` element in `document` (when set); or
    - `undefined` otherwise

    > **Explanation**
    >
    > This value is used in the step on base direction below.

3. (§ 2.6.3.10 Title) if *manifest["name"]* is `undefined`, locate the `title` element [html] using *document* (when set). If that element exists and is non-empty, let *t* be its text content, and add to *manifest*:
    - if the language of `title` is explicitly set to the value of *l*, then add
      `"name": [{"value": t, "language": l}]`
    - otherwise
      `"name": [t]`

    > **Explanation**
    >
    > This step adds the content of the `title` element of *document* when the `name` property is not specified in the manifest. For example:

EXAMPLE 40

```
<html>
<head>
    <title>Moby Dick</title>
    …
    <script type="application/ld+json">
    {
        "@context" : ["https://schema.org","https://www.w3.org/ns/wp-context"],
        …
    }
    </script>
```

yields:

EXAMPLE 41

```
{
    "@context" : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "name"     : ["Moby Dick"],
    …
}
```

4. (§ 2.6.3.6 Language and Base Direction) if *manifest["inLanguage"]* is `undefined` and the value of *lang* is *not* `undefined`, add
`"inLanguage": lang`
to *manifest*

Explanation

This step ensures that a language explicitly set in *document* is valid. For example,

EXAMPLE 42

```
<html>
<head>
    …
    <script type="application/ld+json" lang=ur>
    {
        "@context" : ["https://schema.org","https://www.w3.org/ns/wp-context"],
        …
    }
    </script>
```

yields (unless the language and the direction are set explicitly in the manifest):

EXAMPLE 43

```
{
    "@context"    : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "inLanguage"  : "ur",
    …
}
```

5. (§ 2.6.3.6 Language and Base Direction) if *manifest["inDirection"]* is `undefined` and the value of *dir* is *not* `undefined`, add
`"inDirection": dir`
to *manifest*

Explanation

This step ensures that the base direction explicitly set in the embedding document is valid. For example,

EXAMPLE 44

```
<html>
<head>
    …
    <script type="application/ld+json" dir=rtl lang=ur>
    {
        "@context" : ["https://schema.org","https://www.w3.org/ns/wp-context"],
        "inLanguage"  : "ur",
        …
    }
    </script>
```

yields (unless the language and the direction are set explicitly in the manifest):

EXAMPLE 45

```
{
    "@context"    : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "inLanguage"  : "ur",
    "inDirection" : "rtl"
    …
}
```

6. (§ 2.6.4.1 Default Reading Order) if *manifest["readingOrder"]* is `undefined`, let *u* be the value of document.URL, and add
`"readingOrder": [{"type": ["LinkedResource"], "url": u}]`
to the *manifest*

> **Explanation**
>
> If the Digital Publication consists only of the referencing document, the default reading order can be omitted; it will consist, automatically, of that single resource.

7. (§ 2.6.2.6 Arrays) for each value *v* of *P["term"]* that is a single string or an object, and where *term* expects an array: change the relevant term/value pair to
`"term": [v]`

> **Explanation**
>
> A number of terms require their values to be arrays but, for the sake of convenience, authors are allowed to use a single value instead of a one element array. For example,
>
> EXAMPLE 46
>
> ```
> {
>     "@context"  : ["https://schema.org","https://www.w3.org/ns/wp-context"],
>     "name"      : "Moby Dick",
>     "author"    : "Herman Melville",
>     "resources" : [{
>         "type"           : "LinkedResource",
>         "rel"            : "cover",
>         "url"            : "images/cover.jpg",
>         "encodingFormat" : "image/jpeg"
>     },
>         …
>     }],
>     …
> }
> ```
>
> yields:

EXAMPLE 47

```
{
    "@context"  : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "name"      : ["Moby Dick"],
    "author"    : ["Herman Melville"],
    "resources" : [{
        "type"           : ["LinkedResource"],
        "rel"            : ["cover"],
        "url"            : "images/cover.jpg",
        "encodingFormat" : "image/jpeg"
    },
        …
    }],
    …
}
```

8. (§ 2.6.3.4 Creators) for each value *v* in a *manifest["term"]* array that is a simple string or a `localizable string`, and where *term* expects an entity: exchange that element in the array to `{"type": ["Person"], "name": [v]}`

Explanation

An author, editor, etc., should be explicitly designed as an object of type `Person` but, for the sake of convenience, authors are allowed to just give their name. For example,

EXAMPLE 48

```
{
    "@context"  : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "name"      : ["Moby Dick"],
    "author"    : ["Herman Melville"],
    …
}
```

yields:

EXAMPLE 49

```
{
    "@context"  : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "name"      : ["Moby Dick"],
    "author"    : [{
        "type" : ["Person"],
        "name" : "Herman Melville"
    }],
    …
}
```

9. (§ 2.6.2.3.3 Links) for each value *v* in a *manifest["term"]* array that is a simple string, and where *term* is one of the resource categorization properties: exchange that element in the array to `{"type": ["LinkedResource"], "url": v}`

Explanation

Resource links should be explicitly designed as an object of type `LinkedResource` but, for the sake of convenience, authors are allowed to just give their absolute or relative URL. For example,

EXAMPLE 50

```
{
    "@context"  : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    …
    "resources" : [
        "css/mobydick.css",
        …
    ],
    …
}
```

yields:

EXAMPLE 51

```
{
    "@context"  : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    …
    "resources" : [{
        "type" : ["LinkedResource"],
        "url"  : "css/mobydick.css"
    },
        …
    ],
    …
}
```

10. (§ 2.6.2.3.1 Localizable Strings) for each value *v* of *P["term"]*, or in *P["term"]* in the case the latter is an array, that is a simple string, and *term* expects a localizable string: change the relevant term/value to:

    - if *manifest[inLanguage]* is set to the value of *l* then
      `"term": {"value": v,"language": l}`
    - otherwise
      `"term": {"value": v}`

Explanation

Natural language text values should be explicitly designed as localizable string objects but, for the sake of convenience, authors are allowed to just use a simple string. I.e., if no language information has been provided (via `inLanguage`) in the manifest then, for example,

EXAMPLE 52

```
{
    "@context"  : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "name"      : ["Moby Dick"],
    "author"    : ["Herman Melville"],
    …
}
```

yields:

EXAMPLE 53

```
{
    "@context"  : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "name"      : [{
        "value" : "Moby Dick"
    }],
    "author"    : [{
        "value" : "Herman Melville"
    }],
    …
}
```

If an explicit language has also been provided in the manifest, that language is also added to the localizable string object. For example,

EXAMPLE 54

```
{
    "@context"   : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "inLanguage" : "en",
    "name"       : ["Moby Dick"],
    "author"     : ["Herman Melville"],
    …
}
```

yields:

EXAMPLE 55

```
{
    "@context"  : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "inLanguage": "en",
    "name"      : [{
        "value"    : "Moby Dick",
        "language" : "en"
    }],
    "author"    : [{
        "value"    : "Herman Melville"
        "language" : "en"
    }],
    …
}
```

11. (§ 2.6.2.4 URLs) for each value *v* of *P["term"]* that is not an absolute URL string, and where *term* expects a URL: resolve this value, considered to be a relative URL, using the value of *base* and yielding the value of *au*, and replace the term/value pair by
    `"term": au`

    > **Explanation**
    >
    > All relative URLs in the Publication Manifest must be resolved against the base value to yield absolute URLs.

12. (§ 4.2 Compatibility Requirements, extension point) if a profile defines its own canonicalization steps for profile specific terms, those steps are executed at this point.

13. Return the (transformed) `manifest`.

> NOTE
>
> See the diagram in the appendix for a visual representation of the algorithm. Also, to help understanding the result of the algorithm, there is a link to the corresponding canonical manifests for all the examples in § C. Manifest Examples.

> ISSUE 430 : Should canonicalization include 'absolutization'? priority:high topic:manifest topic:metadata
>
> At the moment, step 11. of the manifest canonicalization means all relative URI-s are resolved at this step using `base`. This may be a problem in relation with the value of `base` in the case of packaged publications, see w3c/pwpub#45.

2.9.4 Post-Processing a Canonical Manifest §

The ***steps for post-processing a canonical manifest*** are given by the following algorithm. The algorithm takes a *json* object representing a canonical manifest. The output from inputting a JSON object into this algorithm is a ***processed manifest***. The goal of the algorithm is to ensure that the data represented in *json* abides to the minimal requirements on the data, removing, if applicable, non-conformant data.

As an abuse of notation, *P["term"]* refers to the value in the object *P* for the label *"term"*, where *P* is either *manifest*, or an object appearing within *manifest* (e.g., a *LinkedResource*).

1. Let *manifest* be the result of converting *json* to a `PublicationManifest` dictionary.

2. Perform data cleanup operations on *manifest*, possibly removing data, as well as raising warnings.

    1. For all *term* that expect entities, check whether the value object *P* in *manifest[term]* has *P["name"]* set. If not, remove *P* from *manifest[term]* array and issue a warning.

    2. For all *term* defined in § 2.6.4 Resource Categorization Properties, check whether every object *P* in *manifest[term]* has *P["url"]* set. If not, remove *P* from *manifest[term]* array and issue a warning. If yes, check whether *P["url"]* is a valid URL [url] and, if not, issue a warning.

3. For every object *P* of type `LinkedResource`, if the value of *P["length"]* is set, check whether this value is a valid number. If the check fails, issue a warning.
4. Check whether the value of *manifest["name"]* is not empty. If it is, generate a value (see the separate note for details) and issue a warning.
5. Check whether *manifest["datePublished"]* is a valid date or date-time, per [iso8601]. If the check fails, issue a warning.
6. Check whether *manifest["dateModified"]* is a valid date or date-time, per [iso8601]. If the check fails, issue a warning.
7. Check whether *manifest["duration"]* is a valid duration value, per [iso8601]. If the check fails, issue a warning.

3. Extension point: process any proprietary, profile specific, and/or other supported members at this point in the algorithm.
4. Return *manifest*.

3. PART II: Web Publications §

3.1 Introduction §

This section is non-normative.

A Web Publication is a discoverable and identifiable collection of resources. Information about the Web Publication is expressed in its manifest, which is an implementation of the format defined in § 2. PART I: Publication Manifest. The manifest is what enables user agents to understand the bounds of the Web Publication and the connection between its resources.

The manifest includes metadata that describe the Web Publication, as a publication has an identity and nature beyond its constituent resources. The manifest also provides a list of all the resources that belong to the Web Publication and a default reading order, which is how it connects resources into a single contiguous work.

A Web Publication is discoverable via the presence of a link to the manifest in any of its resources (i.e., by the use of an HTTP Link header or an HTML `link` element [html]). The linked manifest is contained either directly within the resource containing the link (in a special case called the primary entry page) or in a separate JSON-LD document.

With the establishment of Web Publications, user agents can build new experiences tailored specifically for their unique reading needs.

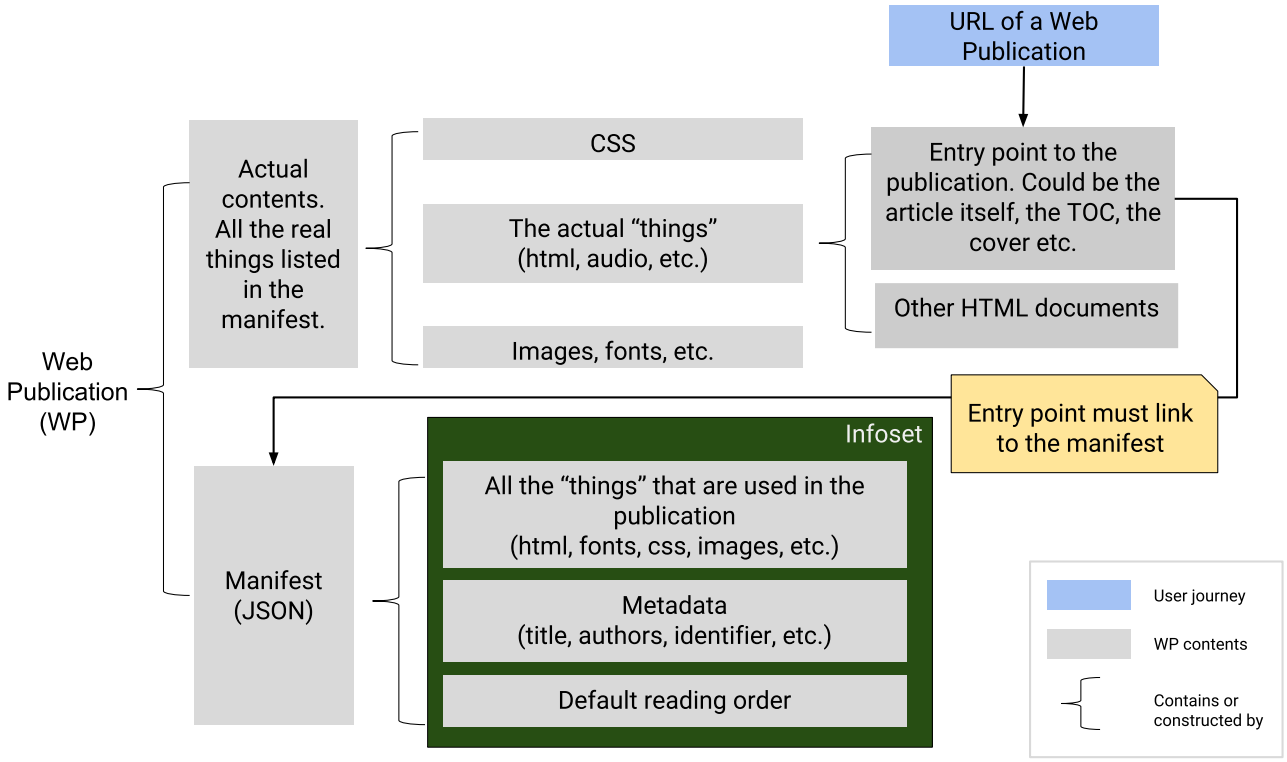

Figure 1 Simplified Diagram of the Structure of Web Publications.
A description of the structure diagram is available in the Appendix. Image available in SVG and PNG formats.

3.2 Conformance Classes §

This specification defines two conformance classes: one for Web Publications and one for user agents that process them.

A Web Publication conforms to this specification if it meets the following criteria:

- it has a manifest that conforms to § 3.4 Manifest;
- it adheres to the construction requirements defined in § 3.3 Web Publication Construction.

A user agent conforms to this specification if it meets the following criteria:

- it is capable of processing a conforming manifest for a Web Publication.

3.3 Web Publication Construction §

3.3.1 Publication Bounds §

A Web Publication consists of a finite set of resources that represent its content. This extent is known as its bounds and is defined within its manifest — it is obtained from the union of resources listed in the default reading order and resource list.

To determine whether a resource is within the bounds of a Web Publication, user agents *MUST* compare the absolute URL of a resource to the absolute URLs of the resources obtained from the union. If the resource is identified in the enumeration, it is within the bounds of the Web Publication. All other resources are external to the Web Publication.

Resources within the bounds of a Web Publication do not have to share the same domain.

3.3.2 Resources §

A Web Publication *MUST* include at least one HTML document [html]—the primary entry page.

Otherwise, a Web Publication *MAY* references resources of any media type, both in the default reading order and as dependencies of other resources.

> NOTE
>
> When adding resources to a Web Publication, consider support in user agents. The use of progressive enhancement techniques and the provision of fallback content, as appropriate, will ensure a more consistent reading experience for users regardless of their preferred user agent.

3.3.3 Primary Entry Page §

The ***primary entry page*** represents the preferred starting resource for a Web Publication and enables discovery of its manifest. It is an [HTML] resource that is returned when accessing the Web Publication's address, and *MUST* be included in either the default reading order or the resource list.

Although any resource can link to the manifest, the primary entry page typically introduces the Web Publication and provides access to the content. It might contain all the content, in the case of a single-page Web Publication, or provide navigational aids to begin reading a multi-document Web Publication. To facilitate the user ease of consumption, the primary entry page *SHOULD* contain the table of contents.

It is not required that the primary entry page be included in the default reading order, nor that it be the first document listed when it is included. This specification leaves the exact nature of the document intentionally underspecified to provide flexibility for different approaches. If a default reading order is not provided, however, user agents will create one using the primary entry page.

The address of the primary entry page is also the canonical identifier for the Web Publication (i.e., it serves as its unique identifier).

In certain cases where information has been omitted from the manifest, user agents will sometimes use the primary entry page as a fallback source of information (see language and base direction and title).

3.3.4 Table of Contents §

The table of contents provides a hierarchical list of links that reflects the structural outline of the major sections of the Web Publication.

The table of contents is expressed via an [html] element (typically a `nav` element) in one of the resources. This element *MUST* be identified by the `role` attribute [html] value "`doc-toc`" [dpub-aria-1.0], and *MUST* be the first element in the document — in document tree order [dom] — with that `role` value.

If the table of contents is not located in the primary entry page, the manifest *SHOULD* identify the resource that contains the structure.

EXAMPLE 56 : If the primary entry page includes the table of contents, no reference to it in the manifest is necessary

```
<head>
    …
    <script type="application/ld+json">
    {
        "@context"          : ["https://schema.org","https://www.w3.org/ns/wp-context"],
        "type"              : "TechArticle",
        …
        "id"                : "http://www.w3.org/TR/tabular-data-model/",
        "url"               : "http://www.w3.org/TR/2015/REC-tabular-data-model-20151217/",
        …
    }
    </script>
    …
</head>
<body>
    …
    <section role="doc-toc">
        …
    </section>
    …
</body>
```

When specified, the table of content *MUST* include a link to at least one resource, and all links *SHOULD* refer to resources within publication bounds.

Refer to the table of contents property definition for more information on how to identify which resource contains the table of contents.

3.3.5 Page List §

The page list is a list of links that provides navigation to static page demarcation points within the content. These locations allow users to coordinate access into the content, but the exact nature of the locations is left to content creators to define. They usually correspond to pages of a print document which is the source of the Web Publication, but might be a purely digital creation added to ease navigation.

The page list is expressed via an [html] element (typically a `nav` element) in one of the resources. This element *MUST* be identified by the `role` attribute [html] value "`doc-pagelist`" [dpub-aria-1.0], and *MUST* be the first element in the document — in document tree order [dom] — with that `role` value.

If the page list is not located in the primary entry page, the manifest *SHOULD* identify the resource that contains the structure.

There are no requirements on the page list itself, except that, when specified, it *MUST* include a link to at least one resource.

Refer to the `pagelist` property definition for more information on how to identify which resource contains the page list.

3.4 Manifest §

3.4.1 Requirements §

The expression of properties in a Web Publication manifest includes a combination of those defined generally for digital publications as well as two specific to Web Publications: the publication's address and canonical identifier.

The requirements for the expression of these properties are as follows:

***REQUIRED*:**

- address
- default reading order
- title

***RECOMMENDED*:**

- accessibility
- base direction
- canonical identifier
- creators
- language and base direction
- links
- last modification date
- publication date
- reading progression direction
- resource list

> NOTE
>
> These properties do not all have to be serialized in the authored manifest. Refer to each property's definition to determine whether it is required in the manifest or can be compiled into the canonical manifest from other information.

In addition, inclusion of the following resources is *RECOMMENDED*:

- accessibility report
- cover
- page list
- privacy policy
- table of contents

3.4.2 Properties §

3.4.2.1 Default Reading Order §

The Web Publication `readingOrder` property extends the Publication Manifest `readingOrder` property in the following ways:

- The default reading order *MAY* be omitted when it only consists of the primary entry page. When the default reading order is absent, user agents *MUST* include an entry for the primary entry page when compiling the canonical manifest.

3.4.2.2 Title §

The Web Publication `title` property extends the Publication Manifest `title` property in the following ways:

- If a title is not included in the authored manifest, the user agent *MUST* use the value of the `title` element [html] of the Web Publication's primary entry page, if present, when generating the canonical manifest.

> NOTE
>
> Relying on the `title` element could be semantically problematic if the Web Publication consists of several HTML resources (e.g., one per chapter of a book), because the HTML definition defines this element as "metadata" for the enclosing HTML document, not for a collection of resources. Using this element is, on the other hand, preferred in the case of a Web Publication consisting of a single HTML document (e.g., a scholarly journal article).

3.5 Association §

3.5.1 Manifest §

The primary entry page *MUST* provide a link to the manifest to enable its discovery. It is the only resource that can provide this link.

The primary entry page is also the only resource in which a manifest can be embedded. It is *RECOMMENDED* to embed the manifest in the primary entry page, but the manifest *MAY* be external to it.

> NOTE
>
> Embedding is the preferred option as search engines might only process schema.org metadata in JSON-LD format when it is embedded in an HTML page.

3.5.2 Publication §

A link to the primary entry page *MAY* be included in any resource to establish that it belongs to a Web Publication.

> EDITOR'S NOTE
>
> Need to determine what relation to use for linking.

Linking resources to their Web Publication is encouraged whenever possible, as it allows user agents to ascertain that a resource belongs to a Web Publication regardless of how the user reaches the resource.

The resources of a Web Publication *MUST NOT* link directly to the manifest; only the primary entry page is permitted to provide such a link.

3.6 Web Publication Lifecycle §

The processing of the Web Publication manifest extends the lifecycle defined in § 2.9 Publication Manifest Lifecycle as described in this section.

> NOTE
>
> See the diagrams in the appendix for a visual representation of the lifecycle algorithm.

3.6.1 Obtaining a manifest §

The ***steps for obtaining a manifest***, starting from the primary entry page, are given by the following algorithm.

1. If the primary entry page does not have the media type `text/html` or `application/xhtml+xml`, terminate this algorithm.
2. From the `Document` of the top-level browsing context of the primary entry page, let *origin* be the `Document`'s origin, and *manifest link* be the first `link` element in tree order in `Document` whose `rel` attribute contains the `publication` token.
3. If *origin* is an [html] opaque origin, terminate this algorithm.

4. If *manifest link* is `null`, terminate this algorithm.
5. If *manifest link*'s `href` attribute's value is the empty string, terminate this algorithm.
6. If the `href` attribute value of *manifest link* is equivalent to [URL] *origin*:
    1. If it has a non-null fragment identifying an identifier *id* in `Document`:
        1. Let *embedded manifest script* be the first `script` element in tree order, whose `id` attribute is equal to *id* and whose `type` attribute is equal to `application/ld+json`.
        2. If *embedded manifest script* is `null`, terminate this algorithm.
        3. Let *text* be the child text content of *embedded manifest script*
        4. Let *base* be the value of baseURI of the `script element`.
    2. Otherwise, terminate this algorithm.

    > **Explanation**
    >
    > This branch is in use when the manifest is embedded in the primary entry page. The algorithm locates the `script` element and extract the manifest itself. The document's URL or, if set by the author, the value of a possible `base` element will be used to turn relative URLs into absolute ones.

7. Otherwise:
    1. Let *manifest URL* be the result of parsing the value of the `href` attribute, relative to the element's base URL. If parsing fails, then abort these steps.
    2. Let *request* be a new [fetch] request, whose URL is *manifest URL*, and whose context is the same as the browsing context of the `Document`.
    3. If the *manifest link*'s `crossOrigin` attribute's value is '`use-credentials`', then set *request*'s credentials to '`include`'.
    4. Await the result of performing a fetch with *request*, letting *response* be the result.
    5. If *response* is a network error, terminate this algorithm.
    6. Let *text* be the result of UTF-8 decoding *response*'s body.
    7. Let *base* be the value of *manifest URL*.

    > **Explanation**
    >
    > This branch is in use when the manifest is in a separate file. It performs the standard operations to retrieve the manifest from the Web; the URL of the manifest file will be used to turn relative URLs into absolute ones.

If `text` contains a non-empty string, it is the input to the first step in the processing stage, with `base` as the base URL, and `Document` as the primary entry page. Otherwise, terminate this algorithm.

3.6.2 Processing a Manifest §

The processing of a Web Publication adds the following requirement to the minimal canonical manifest requirements step:

- if the address is not equivalent to [URL] the URL of *document*, terminate the algorithm.

3.6.3 Extracting a Table of Contents §

If a user agent requires the table of contents, it *MUST* compute the table of contents as follows:

1. Identify the table of contents resource:
    - If a resource in either the default reading order or resource-list is identified with a `rel` value including `contents` [iana-link-relations], the corresponding `url` value identifies the table of contents resource. If there are several such resources, the first one *MUST* be used, with the default reading order taking precedence over resource-list.
    - Otherwise, the primary entry page is the table of contents resource.
2. If the table of contents resource contains an HTML element with the `role` [html] value `doc-toc` [dpub-aria-1.0], the user agent *MUST* use that element as the table of contents. If there are several such HTML elements the user agent *MUST* use the first in document tree order [dom].

See the separate section § B. Machine-Processable Table of Contents for the HTML structure that the table of content *SHOULD* adhere to.

> NOTE
>
> There is no fixed time in the manifest lifecycle when processing of the table of contents has to occur, only that it cannot occur before generating the canonical manifest.

EXAMPLE 57 : Table of content identified in another resource of the publication

```
{
    "@context"   : ["https://schema.org","https://www.w3.org/ns/wp-context"],
    "type"       : "Book",
    …
    "url"        : "https://publisher.example.org/mobydick",
    "name"       : "Moby Dick",
    "resources"  : [{
        "type"       : "LinkedResource",
        "url"        : "toc_file.html",
        "rel"        : "contents"
    },{
        …
    }],
    …
}
```

EXAMPLE 58 : If the primary entry page includes the TOC, no reference in the manifest is necessary

```
<head>
    …
    <script type="application/ld+json">
    {
        "@context"        : ["https://schema.org","https://www.w3.org/ns/wp-context"],
        "type"            : "TechArticle",
        …
        "id"              : "http://www.w3.org/TR/tabular-data-model/",
        "url"             : "http://www.w3.org/TR/2015/REC-tabular-data-model-20151217/",
        …
    }
    </script>
    …
</head>
<body>
    …
    <section role="doc-toc">
        …
    </section>
    …
</body>
```

3.6.4 Extracting a Page List §

User agents *MUST* compute the page list as follows:

1. Identify the page list resource:
    - If a resource in either the default reading order or resource-list is identified with a `rel` value including `pagelist`, the corresponding `url` value identifies the page list resource. If there are several such resources, the first one *MUST* be used, with the default reading order taking precedence over resource-list.
    - Otherwise, the primary entry page is the page list resource.
2. If the page list resource contains an HTML element with the `role` [html] value `doc-pagelist` [dpub-aria-1.0], the user agent *MUST* use that element as the page list. If there are several such HTML elements the user agent *MUST* use the first in document tree order [dom].

EDITOR'S NOTE

The Working Group will attempt to define the `pagelist` term with IANA, to avoid using a URL.

NOTE

There is no fixed time in the manifest lifecycle when processing of the page list has to occur, only that it cannot occur before generating the canonical manifest.

EXAMPLE 59 : Page list identified in another resource of the publication

```
{
"@context"   : ["https://schema.org","https://www.w3.org/ns/wp-context"],
"type"       : "Book",
…
"url"        : "https://publisher.example.org/mobydick",
"name"       : "Moby Dick",
"resources"  : [{
        "type"       : "LinkedResource",
        "url"        : "toc_file.html",
        "rel"        : "pagelist"
},{
        …
}],
…
}
```

3.7 Security §

EDITOR'S NOTE
Placeholder for security issues.

3.8 Privacy §

EDITOR'S NOTE
Placeholder for privacy issues.

4. PART III: Modular Extensions §

4.1 Introduction §

This section is non-normative.

The first two parts of this specification define the cornerstones for developing additional digital publication formats that can be deployed to the Web. Various publishing communities (e.g., audio books, scholarly publications) *MAY* define

extensions, also known as ***profiles***, by *extending* the core § 2. PART I: Publication Manifest with module specific terms and possibly adding new requirements.

4.2 Compatibility Requirements §

In order for a digital publication format to be compatible with this specification, following conditions *MUST* be met:

- It *MUST* adhere to the manifest format requirements defined in § 2. PART I: Publication Manifest.
- The generic canonicalization steps described in § 2.9.3 Generating a Canonical Manifest *MUST* remain valid for the extended manifest. To achieve this, and if new terms are added to the general § 2. PART I: Publication Manifest, then:
  - The term *SHOULD* be categorized, if applicable, to one or more of the general term categories used in the algorithm (e.g., array or localizable string). This means the relevant canonicalization steps will be automatically executed for those terms
  - If necessary, the profile *MAY* define its own canonicalization step, to be executed as the last step of the general canonicalization algorithm. Such an extra step *MUST NOT* invalidate the results of any of the steps defined for the canonicalization algorithm in general.
- If necessary, the profile *MAY* define its own manifest processing step, to be executed as part of the steps described in § 2.9.2 Processing a Manifest

> EDITOR'S NOTE
>
> Adding an example of a term added by, e.g., the audiobook profile would be a good idea, when available.

A. `LinkedResource` Definition §

This specification defines a new type for links called ***`LinkedResource`***. It consists of the following properties:

Term	Description	Required Value	Value Type	[schema.org] Mapping
`url`	Location of the resource. *REQUIRED*.	A valid URL string [url]. Refer to the property definitions that accept this type for additional restrictions.	URL	`url`
`encodingFormat`	Media type of the resource (e.g., `text/html`). *OPTIONAL*.	MIME Media Type [rfc2046].	Literal	`encodingFormat`

Term	Description	Required Value	Value Type	[schema.org] Mapping
name	Name of the item. *OPTIONAL*.	One or more Text items.	Array of Localizable Strings	`name`
description	Description of the item. *OPTIONAL*.	Text.	Localizable String	`description`
rel	The relation of the resource to the publication. *OPTIONAL*.	One or more relations. The values are either the relevant relation terms of the IANA link registry [iana-link-relations], or specially-defined URLs if no suitable link registry item exists.	Array of Literals	(None)
integrity	A cryptographic hashing of the resource that allows its integrity to be verified. *OPTIONAL*.	One or more whitespace-separated sets of integrity metadata [sri]. The value *MUST* conform to the metadata definition [sri]. Refer to [sri] for the list of cryptographic hashing functions that user agents are expected to support.	Literal	(None)
length	The total length of a time-based media resource in (possibly fractional) seconds. *OPTIONAL*	Number	Number	(None)

Although user agent support for the `integrity` property is *OPTIONAL*, user agents that support cryptographic hashing comparisons using this property *MUST* do so in accordance with [sri].

EXAMPLE 60 : A resource with a SHA-256 hashing of its content

```
{
    "type":           "LinkedResource",
    "url":            "chapter1.html",
    "encodingFormat": "text/html",
    "name":           "Chapter 1 - Loomings",
    "integrity":      "sha256-13AE04E21177BABEDFDE721577615A638341F963731EA936BBB8C3862F57CDFC"
}
```

WebIDL

```
dictionary LinkedResource {
    required DOMString           url;
             DOMString           encodingFormat;
             sequence<LocalizableString>   name;
             LocalizableString   description;
             sequence<DOMString> rel;
             DOMString           integrity;
             double              length;
};
```

B. Machine-Processable Table of Contents §

B.1 Introduction §

This section is non-normative.

To facilitate navigation within pages and across sites, HTML uses the `nav` element [html] to express lists of links. Although generic in nature by default, the purpose of a `nav` element can be more specifically identified by use of the `role` attribute [html]. In particular, the `doc-toc` role from the [dpub-aria-1.0] vocabulary identifies the `nav` element as the Web Publications's table of contents.

Including an identifiable table of contents is an accessible way to produce any digital publication, but due to the flexibility of HTML markup, it also presents challenges for user agents trying to extract a meaningful hierarchy of links (e.g., to provide a custom view available from any page). To avoid duplicating the tables of contents for different uses, this section defines a syntax that is both human friendly and commonly used while still providing enough structure for user agent extraction.

Authors have a choice of lists (ordered or unordered) to construct their table of contents. By tagging each link within these lists in anchor tags (`a` elements), user agents can easily differentiate the information they need from any peripheral content (asides) or stylistic tagging that has also been added. The table of contents can consist of both active

links (with an `href` attribute) and inactive links (excluding the `href` attribute), providing additional flexibility in how the table of contents is constructed (e.g., to omit links to certain headings or only link to certain content in a preview).

B.2 HTML Structure §

This section is non-normative.

The machine-readable table of contents is defined within an [html] `nav` element. As described in § 3.3.4 Table of Contents, this `nav` element has to be both the first element in the document and identifiable by a `role` attribute with the value `doc-toc`.

Although the content model of the `nav` element is not restricted, user agents will only be able to extract a usable table of contents when the following markup guidelines are followed:

Table of Contents Title

Although a title for the table of contents is optional, to avoid having a user agent generate a placeholder title when one is needed, it is advised to add one. Titles are specified using any of the [html] `h1` through `h6` elements. Note that only the first such element is recognized as the title. If a heading element is not found before the list of links, user agents will assume that one has not been specified.

List of Links

The first [html] `ol` or `ul` list element encountered in the `nav` element is assumed to contain the list that defines the links into the content. This list will be found even if it is nested inside of `div` elements, for example, as the algorithm ignores elements that are not relevant to its processing. The list cannot occur inside of any skipped elements, however, since their internal contents are not evaluated.

If the `nav` element does not contain one of these elements, then user agents will not register the digital publication as containing a usable table of contents (e.g., a machine-rendered option will not be available).

Branches

If the table of contents is considered as a tree of links, then each list item (`li` element) inside of the list of links represents one branch. Each of these branches has to have a name and optional destination in order to be presented to users, and this information is obtained from the first `a` element found within the list item, wherever it is nested (again, excluding any `a` elements inside of skipped elements.)

The link destination for the branch is obtained from the `a` element's `href` attribute, when specified. This attribute can be omitted if a link is not available (e.g., in a preview) or not relevant (e.g., a grouping header). When providing a link into the content, it is also possible to specify the relation of the linked document (in a `rel` attribute) and the media type of the linked resource (in a `type` attribute).

After finding the `a` element that labels the branch, user agents will continue to inspect the markup for another list element (i.e., sub-branches). If a list is found, it is similarly processed to extract its links, and so on, until there are no more nested branches left to process.

Skipped Elements

A small set of elements are ignored when the parsing table of contents to avoid misinterpretation. These are the [html] sectioning content elements and sectioning root elements. The reason they are ignored is because they can defined their own outlines (i.e., they can represent embedded content that is self-contained and not necessarily related to the structure of content links).

Any element that has its `hidden` attribute set is also skipped, since hidden elements are not intended to be directly accessed by users.

Although these elements can be included in the `nav` element, care has to be taken not to embed important content within them (e.g., do not wrap a `section` element around the list item that contains all the links into the content).

Ignored Elements

All elements that are not relevant to extracting the table of contents, and are not skipped, are ignored. Unlike skipped elements, ignoring means that user agents will continue to search inside them for relevant content, allowing greater flexibility in terms of the tagging that can be used.

B.2.1 Examples §

This section is non-normative.

EXAMPLE 61 : A basic multi-level table of contents.

Note that different list types can be used for the different levels.

```
<nav role="doc-toc">
   <h2>Contents</h2>

   <ol>
      <li>
        <a href="discourses.html">ZARATHUSTRA'S DISCOURSES.</a>
         <ul>
            <li><a href="discourses.html#s01">THE THREE METAMORPHOSES.</a></li>
            <li><a href="discourses.html#s02">THE ACADEMIC CHAIRS OF VIRTUE.</a></li>
            <li><a href="discourses.html#s03">BACKWORLDSMEN.</a></li>
            …
         </ul>
      </li>
      …
   </ol>
</nav>
```

EXAMPLE 62 : A table of contents with ignored content.

The supplementary descriptive information is ignored by user agents.

```
<nav role="doc-toc">
   <h2>Contents</h2>

   <ol>
      <li>
         <div class="title"><a href="c01.html">CHAPTER I</a></div>
         <div class="description">Biographical and Introductory.</div>
      </li>
      <li>
         <div class="title"><a href="c02.html">CHAPTER II</a></div>
         <div class="description">A New System of Alternating Current Motors and Transformers.<
      </li>
      …
   </ol>
</nav>
```

EXAMPLE 63 : A table of contents for a preview.

The `a` elements that link to content the user does not have access to do not include `href` attributes.

```
<nav role="doc-toc">
   <h2>Contents</h2>

  <ol>
     <li><a href="xmas_carol.html">Marley's Ghost</a></li>
     <li><a>The First of Three Spirits</a></li>
     <li><a>The Second of Three Spirits</a></li>
     <li><a>The Last of the Spirits</a></li>
     <li><a>The End of It</a></li>
  </ol>

   …
</nav>
```

EXAMPLE 64 : A table of contents with unlinked headings.

In this example, the author names are not relevant link locations so `href` attributes are not included on their enclosing `a` elements.

```
<nav role="doc-toc">
   <h2>Contents</h2>

   <ol>
      <li>
         <a>Faraday, Michael</a>
         <ol>
            <li><a href="faraday.html#s01">Experimental Researches in Electricity</a></li>
            <li><a href="faraday.html#s02">The Chemical History of a Candle</a></li>
         </ol>
      </li>
      <li>
         <a>Forel, Auguste</a>
         <ol>
            <li><a href="forel.html">The Senses of Insects</a></li>
         </ol>
      </li>
      …
   </ol>
</nav>
```

B.3 User Agent Processing §

This section defines an algorithm for extracting a table of contents from a `nav` element. It is defined in terms of a walk over the nodes of a DOM tree, in tree order, with each node being visited when it is *entered* and when it is *exited* during the walk. Each time a node is visited, it can be seen as triggering an *enter* or *exit* event. In some steps, user agents are provided a choice in how to process the content to provide flexibility for different presentation models.

NOTE

For illustrative purposes, the examples in this section show the structure of the table of contents as JavaScript objects. User agents can process and internalize the resulting structure in whatever language and form is appropriate.

For the purposes of this algorithm, a ***list element*** is defined as either an [html] `ol` or `ul` element.

The following algorithm *MUST* be applied to a walk of a DOM subtree rooted at the first `nav` element in document order with the `role` attribute value `doc-toc`. All explanations are *informative*.

1. Let *toc* be a object that represents the table of contents and initialize it as follows:
    1. Create a `name` property for *toc* that represents the title of the table of contents and set to an empty string.

2. Create an `entries` property for *toc* that represents all the branches of the table of contents and set to an empty array.

> **Explanation**
>
> This step initializes the *toc* object that will store the title and the branches of the table of contents.
>
> EXAMPLE 65 : Visualization of the default toc object
>
> ```
> {
>    "name": '',
>    "entries": []
> }
> ```

2. Initialize a stack.

> **Explanation**
>
> The stack is used to hold branches that are not yet complete. As a new sub-branch is encountered, the parent gets pushed onto the stack so it can be retrieved later.

3. Let *current toc branch* be a variable set to `null`.

> **Explanation**
>
> *current toc branch* is used to hold the object that represents the branch of the table of contents that is currently being processed.

4. Walk over the DOM in tree order, starting with the `nav` element the table of contents is being built from, and trigger the first relevant step below for each element as the walk enters and exits it.

    1. **When entering a heading content element:**

        Run these steps:

        1. If the stack is empty, and the `name` property of *toc* is an empty string, set the `name` property to one of the following:

            - the descendant content of the element (to preserve any HTML tags);
            - the text string obtained from the descendant content (e.g., by calculating the accessible name [accname-1.1] of the element).

            If the resulting value of `name` is an empty string (e.g., after removing any presentational elements and trimming all leading and trailing whitespace), set the `name` property either to a placeholder value or to `null`.

        2. Exit the element and continue processing with the next element.

Explanation

This step identifies the heading for the table of contents. A heading is only processed if the value of the *toc* `name` property is an empty string (i.e., no headings have yet been encountered).

Whether a user agent sets the `name` to the descendant content of the heading element, or generates a text string from it, depends on whether it will re-use any descendant tagging in the presentation (e.g., to retain images, MathML, ruby and other content that does not translate to text easily).

EXAMPLE 66 : Visualization of the toc object with a heading

```
{
   "name": "Contents",
   "entries": []
}
```

If the `name` is not an empty string, or is `null`, then a previous heading has already been encountered or content has been encountered that indicates the `nav` element does not have a heading (e.g., a list has already been processed, since the heading would not follow the list of links).

EXAMPLE 67 : Visualization of the toc object without a heading

```
{
   "name": null,
   "entries": []
}
```

If a heading is not specified, the user agent can provide its own for later use.

2. **When entering a list element:**

   Run these steps:

   1. If the `name` property of *toc* is an empty string, set `name` to `null`.
   2. If *current toc branch* is not `null`:
      1. If the `entries` property of *current toc branch* is `null` or a non-empty array, exit the element and continue processing with the next element.
      2. Otherwise, push the object in *current toc branch* onto the stack and set *current toc branch* to `null`.
   3. Otherwise, if the stack is empty:
      1. If the `entries` property of *toc* is `null` or a non-empty array, exit the element and continue processing with the next element.
      2. Otherwise, do nothing.

> **Explanation**
>
> This algorithm does not process multiple lists in a single branch or at the root of the `nav` element, so if a list has already been encountered (the `entries` property contains one or more branches or is set to `null`), this list is skipped.
>
> If a list is encountered and the table of contents (`toc`) still does not have a name (i.e., no heading element has been encountered), the table of contents is assumed to not have a heading (i.e., the heading for the table of contents cannot appear after the first list of entries). The value of the `name` property is changed from an empty string to `null` as no further headings encountered apply, either.

3. **When exiting a list element:**

   If the stack is not empty, pop the top object off the stack and set *current toc branch* to it.

   > **Explanation**
   >
   > This resets *current toc branch* back to the parent object after all of its child branches have been processed.

4. **When entering a list item element:**

   Run these steps:

   1. Set *current toc branch* to a new object.
   2. Create `name`, `url`, `type`, and `rel` properties for the object and set them to empty strings.
   3. Create an `entries` property for the object and set it to an empty array.

   > **Explanation**
   >
   > Each list item represents a possible new branch in the table of contents, so whenever one is encountered a new blank object is created in *current toc branch*.
   >
   > EXAMPLE 68 : Visualization of a new branch object
   >
   > ```
   > {
   >    "name": '',
   >    "url": '',
   >    "type": '',
   >    "rel": '',
   >    "entries": []
   > }
   > ```
   >
   > This object gets populated with information as a descendant `a` element and list are encountered.

5. **When exiting a list item element:**

   Run these steps:

1. If `entries` property of *current toc branch* contains an empty array, set its value to `null`.
2. If the stack contains one or more entries:
    1. If the `entries` property of *current toc branch* contains a non-empty array, and its `name` property is an empty string, set its `name` to a placeholder value or `null`;
    2. If the `entries` property of *current toc branch* contains an empty array, and its `name` property is an empty string, set *current toc branch* to null and exit this processing step.

    Add *current toc branch* to the array in the `entries` property of the object at the top of the stack.
3. Otherwise, add the object in *current toc branch* to the `entries` array of *toc*.
4. Set *current toc branch* to `null`.

> **Explanation**
>
> Exiting a list item indicates that processing of the current branch is complete. Before adding this branch to its parent's `entries` array, the branch needs to be tested to see if it has a name and/or any sub-branches. If it does not have a name but has sub-branches, the branch is kept. The user agent can either supply a placeholder value of its own creation or set the value to null. If it does not have a name or any branches, it is invalid and is discarded.
>
> To determine where to merge the branch, the stack is checked. If there are no objects in the stack, it is added into the `entries` property of the root *toc* object (i.e., it is a top-level branch). Otherwise, it gets added into the `entries` property of the object immediately preceding it in the stack.
>
> As a final step, *current toc branch* is reset back to `null`.

EXAMPLE 69 : Visualization of a branch merge

If the following two objects are in *current toc branch*

```
{
   "name": "Section 1",
   "url": "http://example.com/contents.html#s1",
   "type": "text/html",
   "rel": null,
   "entries": []
},
{
   "name": "Section 1.1",
   "url": "http://example.com/contents.html#s1.1",
   "type": "text/html",
   "rel": null,
   "entries": null
}
```

Then only the following single object remains after merging:

```
{
   "name": "Section 1",
   "url": "http://example.com/contents.html#s1",
   "type": "text/html",
   "rel": null,
   "entries": [
      {
         "name": "Section 1.1",
         "url": "http://example.com/contents.html#s1.1",
         "type": "text/html",
         "rel": null,
         "entries": null
      }
   ]
}
```

6. **When entering an anchor element and *current toc branch* is not `null`:**

   Run these steps:

   1. If the `name` property of *current toc branch* is not an empty string, do nothing.
   2. Otherwise:
      1. Set the `name` property of *current toc branch* to one of the following:
         - the descendant content of the anchor element (to preserve any HTML tags);

- the text string obtained from the descendant content (e.g., by calculating the accessible name [accname-1.1] of the element).

If the resulting value of `name` is an empty string (e.g., after removing any presentational elements and trimming all leading and trailing whitespace), set the `name` property to `null`.

2. If the element has an `href` attribute and the URL in the attribute resolves to a resource in the default reading order or resource list, set the `url` property of *current toc branch* to the value. Otherwise, set the property to `null`.
3. If the element has a `type` attribute, and the value of the attribute is not an empty string after trimming leading and trailing white space, set the `type` property of *current toc branch* to its value. Otherwise, set the property to `null`.
4. If the element has a `rel` attribute, and the value of the attribute is not an empty string after trimming leading and trailing white space, set the `rel` property of *current toc branch* to its value. Otherwise, set the property to `null`.

Exit the element and continue processing with the next element.

Explanation

This step processes anchor tags to obtain values for the `name` and `url` properties of a branch.

If the name of the current branch is already defined, then processing of this element is terminated (i.e., to avoid processing multiple links for a single branch).

Whether a user agent sets the `name` of the entry to the descendant content of the `a` element, or generates a text string from it, depends on whether it will re-use any descendant tagging in the presentation (e.g., to retain images, MathML, ruby and other content that does not translate to text easily).

In addition to having an `href` attribute specified, it is necessary that it resolve to a resource that belongs to the digital publication to meet the requirements of this specification. If not, the branch is retained but the entry will not be linkable.

Additional information about the target of the link — the type of resource and its relation — is also retained.

EXAMPLE 70 : Visualization of a link to an SVG image

```
{
   "name": "In the Beginning",
   "url": "http://example.com/page1.svg",
   "type": "image/svg",
   "rel": null,
   "entries": []
},
```

7. **When entering a sectioning content element, a sectioning root element, or an element with a hidden attribute:**

Exit the element and continue processing with the next element.

> **Explanation**
>
> As sectioning and sectioning root elements can define their own outlines, descending into them poses problems for generating the table of contents (i.e., they may contain content that is not directly related). As a result, they are skipped over when encountered to prevent their child content from being processed.

8. **Otherwise: do nothing.**

> **Explanation**
>
> For all other elements, this steps allows their descendant elements to continue to be processed.

5. After completing the DOM walk, if the *entries* property of *toc* contains a non-empty array, *toc* represents the machine-processed table of contents.

   Otherwise, the digital publication does not have a table of contents that can be used for machine rendering purposes.

   > **Explanation**
   >
   > If the `entries` array in the root *toc* object does not contain any branches (either because no list was found in the `nav` element or the list did not contain any conforming list items), then the algorithm did not produce a usable table of contents.

C. Manifest Examples §

This section is non-normative.

C.1 Simple Book §

A manifest for a simple book. The canonical version of this manifest is also available.

EXAMPLE 71

```
{
    "@context": ["https://schema.org", "https://www.w3.org/ns/wp-context"],
    "type": "Book",
    "url": "https://publisher.example.org/mobydick",
    "author": "Herman Melville",
    "dateModified": "2018-02-10T17:00:00Z",

    "readingOrder": [
        "html/title.html",
        "html/copyright.html",
        "html/introduction.html",
        "html/epigraph.html",
        "html/c001.html",
        "html/c002.html",
        "html/c003.html",
        "html/c004.html",
        "html/c005.html",
        "html/c006.html"
    ],

    "resources": [
        "css/mobydick.css",
        {
            "type": "LinkedResource",
            "rel": "cover",
            "url": "images/cover.jpg",
            "encodingFormat": "image/jpeg"
        },{
            "type": "LinkedResource",
            "url": "html/toc.html",
            "rel": "contents"
        },{
            "type": "LinkedResource",
            "url": "fonts/STIXGeneral.otf",
            "encodingFormat": "application/vnd.ms-opentype"
        },{
            "type": "LinkedResource",
            "url": "fonts/STIXGeneralBol.otf",
            "encodingFormat": "application/vnd.ms-opentype"
        },{
            "type": "LinkedResource",
            "url": "fonts/STIXGeneralBolIta.otf",
            "encodingFormat": "application/vnd.ms-opentype"
        },{
            "type": "LinkedResource",
            "url": "fonts/STIXGeneralItalic.otf",
            "encodingFormat": "application/vnd.ms-opentype"
        }
```

```
        ]
    }
```

C.2 Single-Document Publication §

Example for an embedded manifest example. The canonical version of the manifest is, as well as a more elaborate version for the same document are also available.

EXAMPLE 72

```
<!DOCTYPE html>
<html lang="en-US">
<head>
    <title>Model for Tabular Data and Metadata on the Web</title>
    <link href="#wpm" rel="publication" />
    ...
    <script id="wpm" type="application/ld+json">
    {
        "@context"        : ["https://schema.org", "https://www.w3.org/ns/wp-context"],
        "type"            : "TechArticle",
        "id"              : "http://www.w3.org/TR/tabular-data-model/",
        "url"             : "http://www.w3.org/TR/2015/REC-tabular-data-model-20151217/",
        "copyrightYear"   : "2015",
        "copyrightHolder" : "World Wide Web Consortium",
        "creator"         : ["Jeni Tennison", "Gregg Kellogg", "Ivan Herman"],
        "publisher" : {
            "type" : "Organization",
            "name" : "World Wide Web Consortium",
            "id"   : "https://www.w3.org/"
        },
        "datePublished"         : "2015-12-17",
        "resources"             : [
            "datatypes.html",
            "datatypes.svg",
            "datatypes.png",
            "diff.html",
            {
                "type"           : "LinkedResource",
                "url"            : "test-utf8.csv",
                "encodingFormat" : "text/csv"

            },
            {
                "type"           : "LinkedResource",
                "url"            : "test.xlsx",
                "encodingFormat" : "application/vnd.openxmlformats-officedocument.spreadsheetml
            }
        ],
    }
    </script>
</head>
<body>
    ....

    <section id="toc" role="doc-toc">
        <h2 resource="#h-toc" id="h-toc" class="introductory">Table of Contents</h2>
        <ul class="toc">
            <li class="tocline"><a class="tocxref" href="#intro">
                <span class="secno">1. </span>Introduction</a>
```

```
            </li>
            ...
        </ul>
    </section>
    ...

</body>
</html>
```

C.3 Audiobook §

A manifest for an audiobook. The canonical version of this manifest is also available.

EXAMPLE 73

```
{
  "@context": ["https://schema.org", "https://www.w3.org/ns/wp-context"],
  "type": "Audiobook",
  "id": "https://librivox.org/flatland-a-romance-of-many-dimensions-by-edwin-abbott-abbott/",
  "url": "https://w3c.github.io/wpub/experiments/audiobook/",
  "name": "Flatland: A Romance of Many Dimensions",
  "author": "Edwin Abbott Abbott",
  "readBy": "Ruth Golding",
  "publisher": "Librivox",
  "inLanguage": "en",
  "dateModified": "2018-06-14T19:32:18Z",
  "datePublished": "2008-10-12",
  "duration": "PT15153S",
  "license": "https://creativecommons.org/publicdomain/zero/1.0/",

  "resources": [
    {
      "rel": "cover",
      "url": "http://ia800704.us.archive.org/9/items/LibrivoxCdCoverArt12/Flatland_1109.jpg",
      "encodingFormat": "image/jpeg"
    },{
      "rel": "contents",
      "url": "toc.html",
      "encodingFormat": "text/html"
    }
  ],

  "readingOrder": [
    {
      "url": "http://www.archive.org/download/flatland_rg_librivox/flatland_1_abbott.mp3",
      "encodingFormat": "audio/mpeg",
      "length": 1371,
      "name": "Part 1, Sections 1 - 3"
    },{
      "url": "http://www.archive.org/download/flatland_rg_librivox/flatland_2_abbott.mp3",
      "encodingFormat": "audio/mpeg",
      "length": 1669,
      "name": "Part 1, Sections 4 - 5"
    },{
      "url": "http://www.archive.org/download/flatland_rg_librivox/flatland_3_abbott.mp3",
      "encodingFormat": "audio/mpeg",
      "length": 1506,
      "name": "Part 1, Sections 6 - 7"
    },{
      "url": "http://www.archive.org/download/flatland_rg_librivox/flatland_4_abbott.mp3",
      "encodingFormat": "audio/mpeg",
      "length": 1669,
      "name": "Part 1, Sections 8 - 10"
    },{
```

```
      "url": "http://www.archive.org/download/flatland_rg_librivox/flatland_5_abbott.mp3",
      "encodingFormat": "audio/mpeg",
      "length": 1506,
      "name": "Part 1, Sections 11 - 12"
    },{
      "url": "http://www.archive.org/download/flatland_rg_librivox/flatland_6_abbott.mp3",
      "encodingFormat": "audio/mpeg",
      "length": 1798,
      "name": "Part 2, Sections 13 - 14"
    },{
      "url": "http://www.archive.org/download/flatland_rg_librivox/flatland_7_abbott.mp3",
      "encodingFormat": "audio/mpeg",
      "length": 1225,
      "name": "Part 2, Sections 15 - 17"
    },{
      "url": "http://www.archive.org/download/flatland_rg_librivox/flatland_8_abbott.mp3",
      "encodingFormat": "audio/mpeg",
      "length": 1371,
      "name": "Part 2, Sections 18 - 20"
    },{
      "url": "http://www.archive.org/download/flatland_rg_librivox/flatland_9_abbott.mp3",
      "encodingFormat": "audio/mpeg",
      "length": 1659,
      "name": "Part 2, Sections 21 - 22"
    }
  ]
}
```

D. Examples for bidirectional texts §

This section is non-normative.

This section illustrates how Unicode formatting characters can be applied to bidirectional strings, where necessary, in order to help a consumer produce the expected display. In cases where the first-strong heuristics would produce the wrong result, if the string is created with a prepended formatting character, the first-strong heuristics will produce the correct base direction for the string as a whole.

A right-to-left string that begins with a Latin script character should have U+200F RIGHT-TO-LEFT MARK prepended.

Character order in memory:	HTML איה תפש סימון.
Gives incorrect display:	HTML היא שפת סימון.
Source code with formatting character:	"\u200FHTML איה תפש סימון."
Gives expected display:	היא שפת סימון HTML.

A left-to-right string that begins with a Arabic script character should have U+200E LEFT-TO-RIGHT MARK prepended.

Character order in memory:	`'سلام' is hello in Persian.`
Gives incorrect display:	.is hello in Persian 'سلام'
Source code with formatting character:	`"\u200E'سلام' is hello in Persian."`
Gives expected display:	'سلام' is hello in Persian.

E. Lifecycle diagrams §

This section is non-normative.

These diagrams provide a visual view of the lifecycle steps, as specified in § 3.6 Web Publication Lifecycle.

E.1 Overview of the lifecyle algorithm §

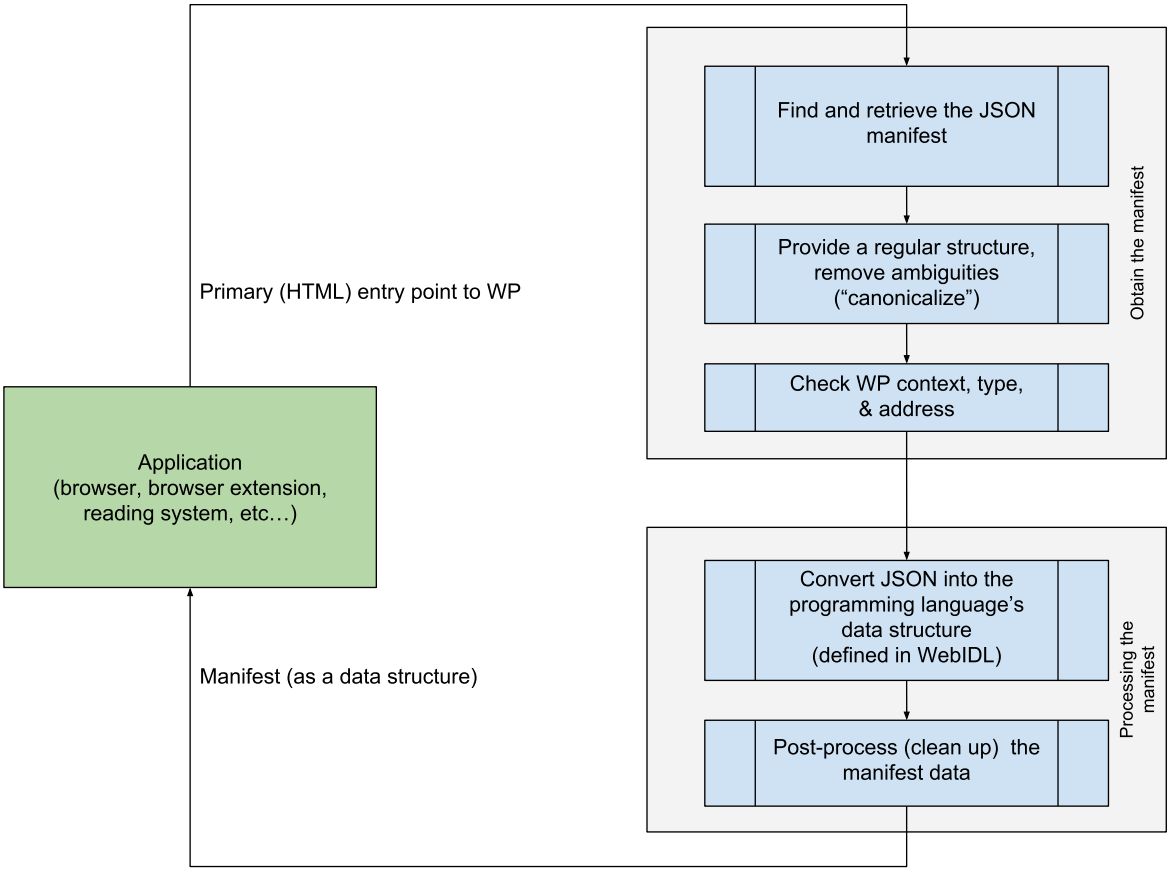

Figure 2 Overview of the lifecyle algorithm, depicting the main building blocks.
See the normative description of the algorithm in § 3.6 Web Publication Lifecycle. Image available in SVG and PNG formats.

E.2 Finding the manifest §

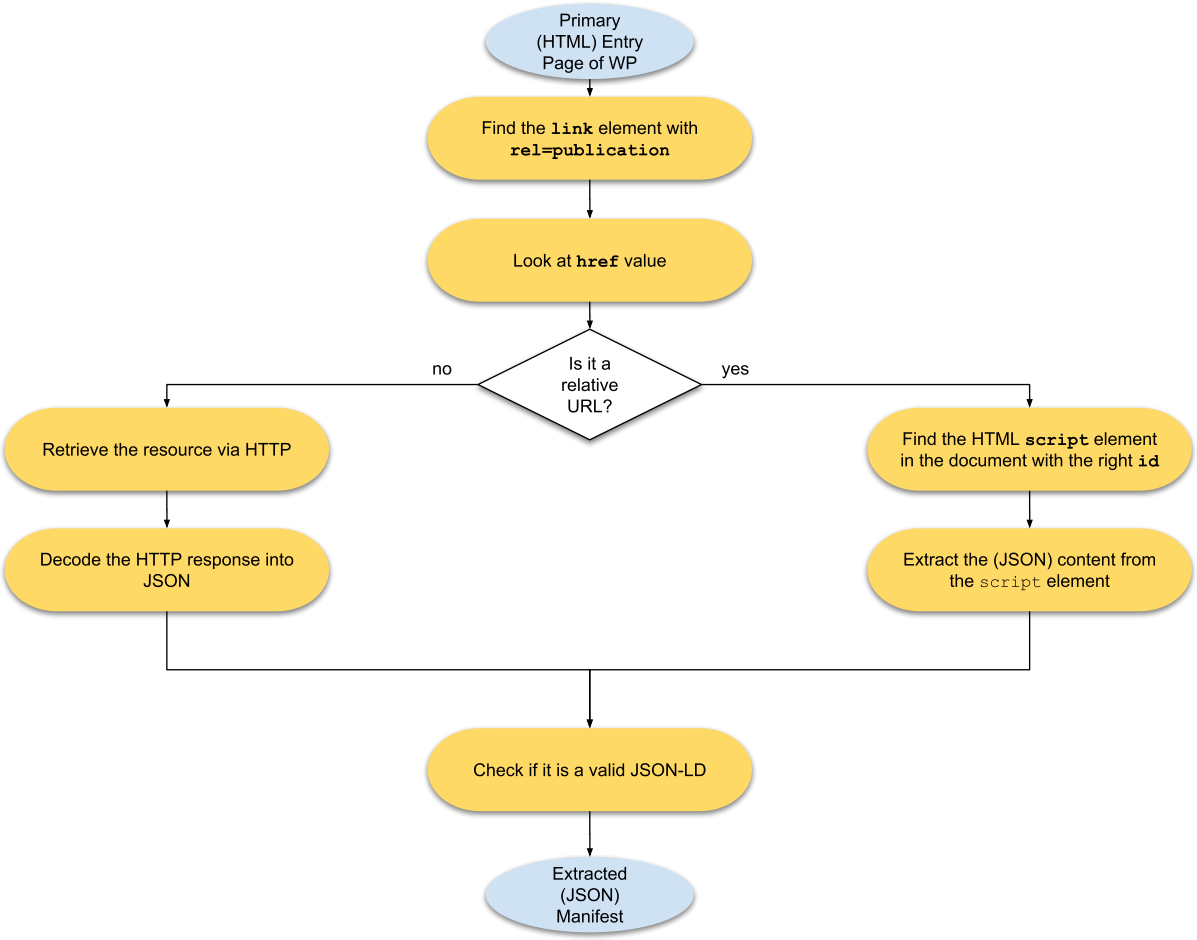

Figure 3 First major block in the lifecyle algorithm: find the manifest, either through an HTTP request or as part of a script elements.

See the normative description of the algorithm in § 2.9.2 Processing a Manifest. Image available in SVG and PNG formats.

E.3 Manifest canonicalization §

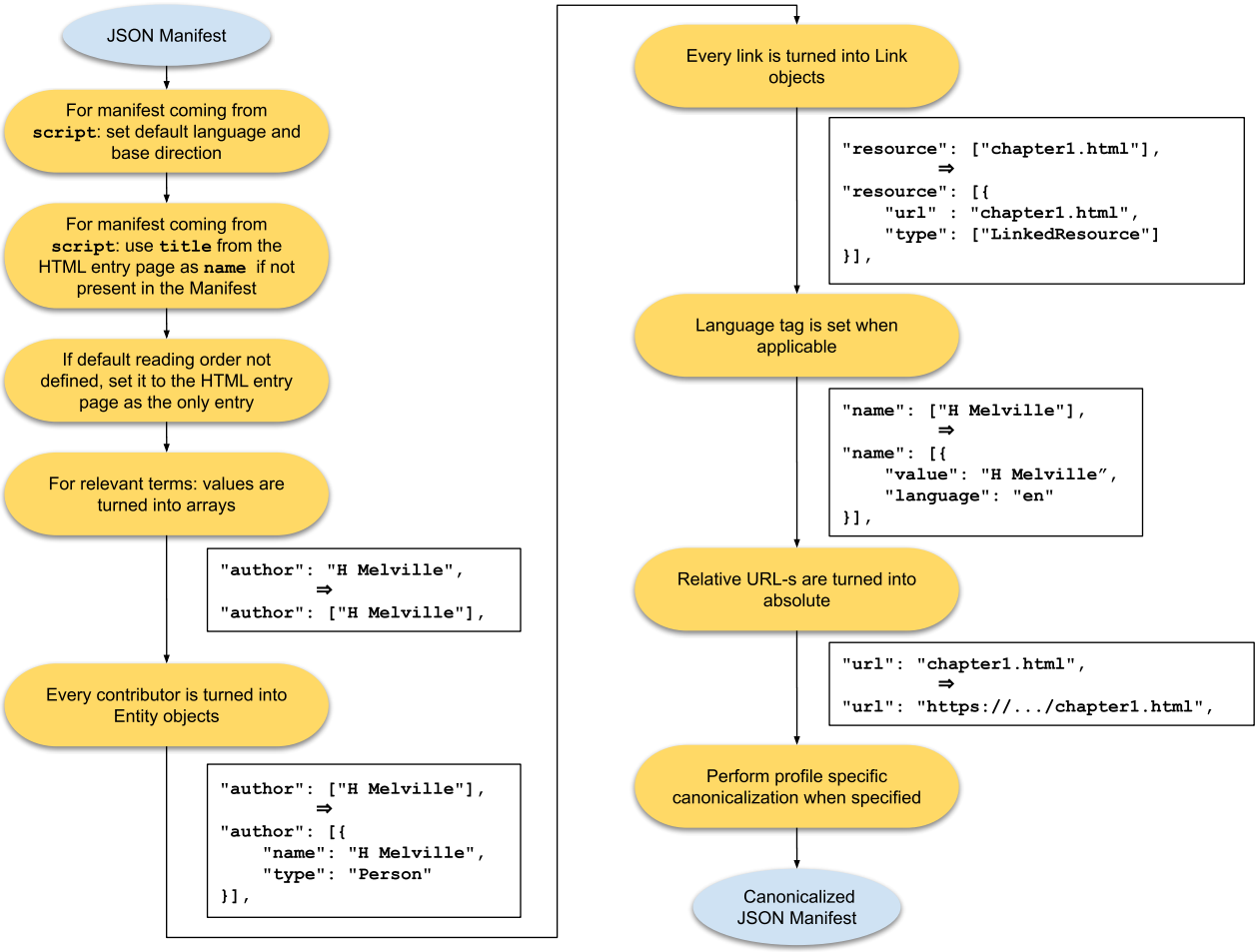

Figure 4 Second major block in the lifecyle algorithm: create a canonical manifest (using the core manifest terms as examples). See the normative description of the algorithm in § 2.9.3 Generating a Canonical Manifest. Image available in SVG and PNG formats.

E.4 Converting the manifest into a data structure §

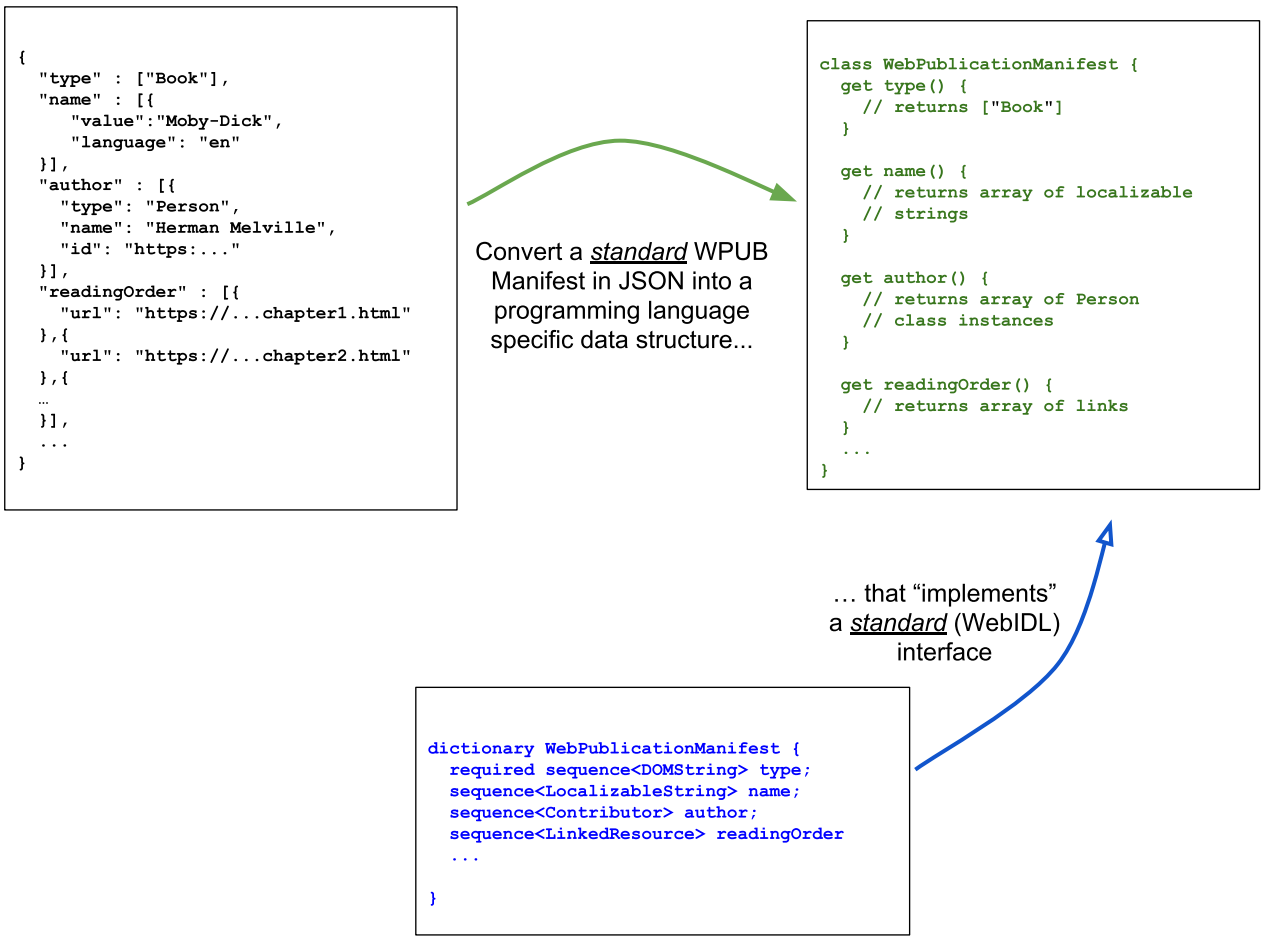

Figure 5 Third major block in the lifecyle algorithm: convert the manifest into a programming language dependent data structure that implements the Web IDL specification of the manifest.

See the normative description of the algorithm in § 2.9.4 Post-Processing a Canonical Manifest. Image available in SVG and PNG formats.

E.5 Cleaning up the data §

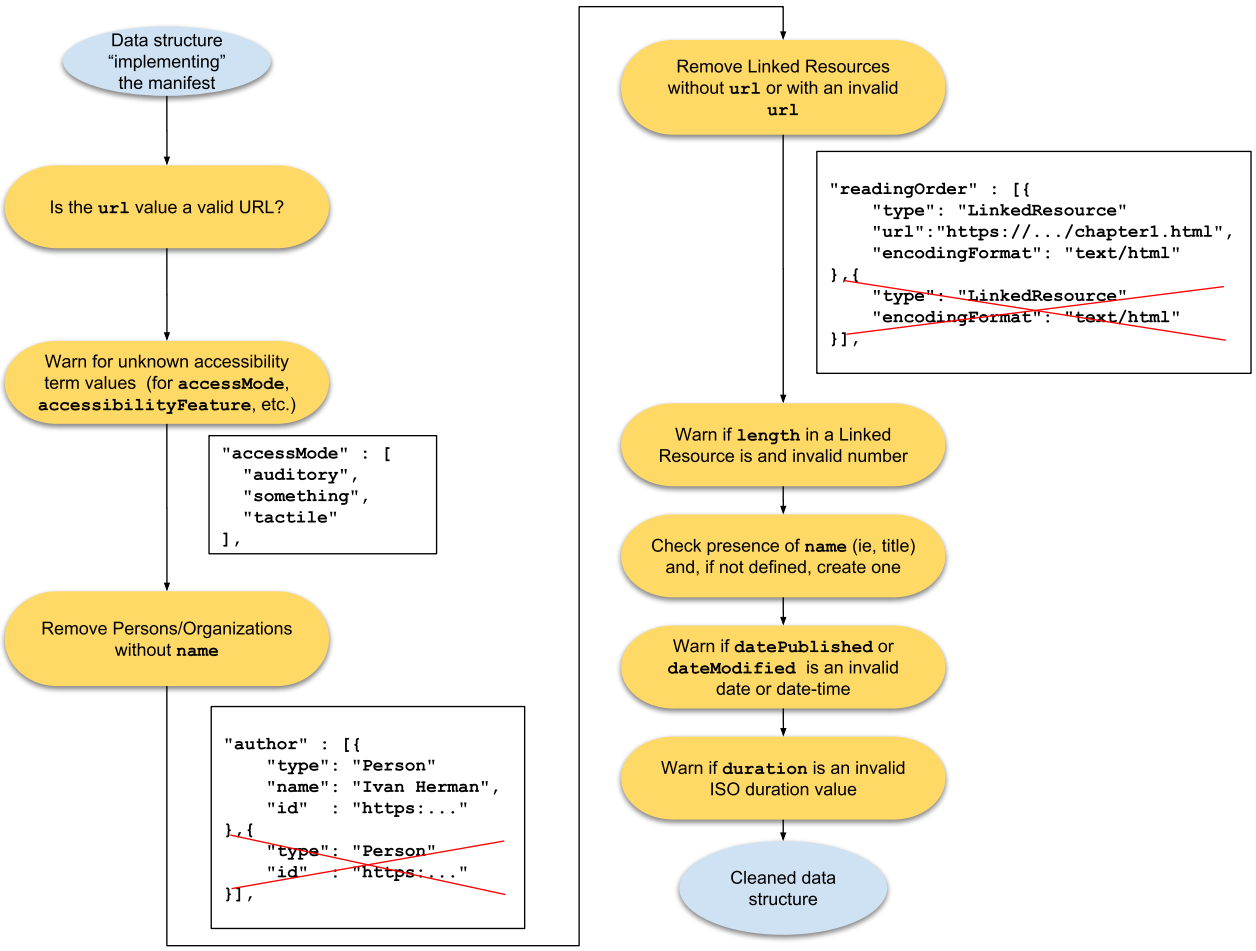

Figure 6 Fourth major block in the lifecyle algorithm: check and clean up data by possibly removing data that cannot be interpreted. See the normative description of the algorithm in § 2.9.4 Post-Processing a Canonical Manifest. Image available in SVG and PNG formats.

F. Properties Index §

This section is non-normative.

The following table identifies where the use of manifest properties is defined and extended.

Name	Publication Manifest	Web Publication
accessMode	§ 2.6.3.1 Accessibility	
accessModeSufficient	§ 2.6.3.1 Accessibility	
accessibilityFeature	§ 2.6.3.1 Accessibility	
accessibilityHazard	§ 2.6.3.1 Accessibility	

Name	Publication Manifest	Web Publication
accessibilitySummary	§ 2.6.3.1 Accessibility	
address		§ 2.6.3.2 Address
artist	§ 2.6.3.4 Creators	
author	§ 2.6.3.4 Creators	
contributor	§ 2.6.3.4 Creators	
creator	§ 2.6.3.4 Creators	
dateModified	§ 2.6.3.7 Last Modification Date	
datePublished	§ 2.6.3.8 Publication Date	
duration	§ 2.6.3.5 Duration	
editor	§ 2.6.3.4 Creators	
id		§ 2.6.3.3 Canonical Identifier
illustrator	§ 2.6.3.4 Creators	
inDirection	§ 2.6.3.6 Language and Base Direction	
inker	§ 2.6.3.4 Creators	
inLanguage	§ 2.6.3.6 Language and Base Direction	
letterer	§ 2.6.3.4 Creators	
link	§ 2.6.4.3 Links	
name	§ 2.6.3.10 Title	Extended in § 3.4.2.2 Title
penciler	§ 2.6.3.4 Creators	
publisher	§ 2.6.3.4 Creators	
readBy	§ 2.6.3.4 Creators	
readingOrder	§ 2.6.4.1 Default Reading Order	Extended in § 3.4.2.1 Default Reading Order
readingProgression	§ 2.6.3.9 Reading Progression Direction	
resources	§ 2.6.4.2 Resource List	
translator	§ 2.6.3.4 Creators	

G. Resource Relations Index §

This section is non-normative.

The following table identifies where the use of resource relations is defined.

Name	Publication Manifest
`accessibility-report`	§ 2.7.2.1 Accessibility Report
`contents`	§ 2.7.3.3 Table of Contents
`cover`	§ 2.7.3.1 Cover
`pagelist`	§ 2.7.3.2 Page List
`privacy-policy`	§ 2.7.2.3 Privacy Policy
`preview`	§ 2.7.2.2 Preview

H. Image Descriptions §

This section is non-normative.

Description for the "Structure of Web Publications" diagram:

A simplified diagram of the structure of a Web Publication. The Web Publication is broken down into two elements. The first element is the actual contents (all the real things listed in the manifest). This element is broken down into the CSS, the actual "things" such as the HTML documents, audio, etc, and the images, fonts etc. The actual "things" have an additional subset of items that includes the entry page to the publication and all of the other documents. The second element is the Manifest (JSON). The manifest is used to generate the canonical manifest, which consists of a list of all the "things" in the publication, the publication metadata, and the default reading order of content. It is noted in the diagram that the entry page has to link to the manifest. (Return to the diagram of Web Publication.)

I. Acknowledgements §

This section is non-normative.

The editors would like to thank the members of the Publishing Working Group for their contributions to this specification:

Greg Albers (J. Paul Getty Trust), Franco Alvarado (Macmillan Learning), Boris Anthony (The Rebus Foundation), Luc Audrain (Hachette Livre), Baldur Bjarnason (The Rebus Foundation), Laura Brady (W3C Invited Expert), Steve Breault (Scenarex Inc.), Don Brutzman (Web3D Consortium), Kaylin Bugbee (Earth Science Data Systems Program), Yu-Wei Chang (Taiwan Digital Publishing Forum), Fred Chasen (W3C Invited Expert), Timothy Cole (University of Illinois at Urbana-Champaign), Simon Collinson (Rakuten, Inc.), Rachel Comerford (Macmillan Learning), Garth Conboy (Google, Inc., chair), Juan Corona (Evident Point Software), Christopher Cosner (Stanford University), Dave Cramer (Hachette Livre), Greg Davis (Pearson plc), Romain Deltour (DAISY Consortium), Marisa DeMeglio (DAISY Consortium), Vagner Diniz (NIC.br - Brazilian Network Information Center), Kenneth Dougherty (Pearson plc), Brady Duga (Google, Inc.), Ben Dugas (Rakuten, Inc.), Roger Espinosa (University of Michigan), Reinaldo Ferraz (NIC.br - Brazilian Network Information Center), Heather Flanagan (RFC Editor), Teenya Franklin (Pearson plc), Jun Gamo (Voyager Japan,

Inc.), Michael Goodman (Wiley), Markku Hakkinen (Educational Testing Service), Katie Haritos-Shea (Knowbility), Geoff Jukes (Blackstone Audio, Inc.), Deborah Kaplan (W3C Invited Expert), Bill Kasdorf (Book Industry Study Group), George Kerscher (DAISY Consortium), Yuri Khramov (Evident Point Software), Masakazu Kitahara (Voyager Japan, Inc.), Toshiaki Koike (Voyager Japan, Inc.), Charles LaPierre (Benetech), Mustapha Lazrek (Microsoft Corp.), Laurent Le Meur (EDRLab), Vladimir Levantovsky (Monotype), Mia Lipner (Pearson plc), Phil Madans (Hachette Livre), Christopher Maden (University of Illinois at Urbana-Champaign), Dmitry Markushevich (Evident Point Software), keith mcfarland (Blackstone Audio, Inc.), Jonathan McGlone (University of Michigan), Hugh McGuire (The Rebus Foundation), Nellie McKesson (W3C Invited Expert), Selma Morais (NIC.br - Brazilian Network Information Center), Jasmine Mulliken (Stanford University), Cristina Mussinelli (Fondazione LIA), Christos Nikolakakos (Wiley), Gregorio Pellegrino (Fondazione LIA), Fernando Pinto da Silva (EDRLab), Nicholas Polys (Web3D Consortium), Chris Powell (University of Michigan), Jeff Printy (Macmillan Learning), Ryan Pugatch (Hachette Livre), Joshua Pyle (Wiley), Wendy Reid (Rakuten, Inc., chair), Florian Rivoal (W3C Invited Expert), Leonard Rosenthol (Adobe), Robert Sanderson (J. Paul Getty Trust), Jodi Schneider (University of Illinois at Urbana-Champaign), Ben Schroeter (Pearson plc), Tzviya Siegman (Wiley, chair), Avneesh Singh (DAISY Consortium), Adam Sisco (Earth Science Data Systems Program), David Stroup (Pearson plc), Mateus Teixeira (W. W. Norton & Company), Jonathan Thurston (Pearson plc), Yukio Tomikura (Kodansha, Publishers, Ltd.), Ben Walters (Microsoft Corp.), Daniel Weck (EDRLab, DAISY Consortium), John Weise (University of Michigan), Jason White (Educational Testing Service), Richard Wright (EDRLab), Jeff Xu (Rakuten, Inc.), Evan Yamanishi (W. W. Norton & Company), Maurice York (University of Michigan), Junichi Yoshii (Kodansha, Publishers, Ltd.), Benjamin Young (Wiley), Mohamed ZERGAOUI (INNOVIMAX)

The Working Group would also like to thank the members of the Digital Publishing Interest Group for all the hard work they did paving the road for this specification.

J. References §

J.1 Normative references §

[accname-1.1]

Accessible Name and Description Computation 1.1. Joanmarie Diggs; Bryan Garaventa; Michael Cooper. W3C. 18 December 2018. W3C Recommendation. URL: https://www.w3.org/TR/accname-1.1/

[bcp47]

Tags for Identifying Languages. A. Phillips; M. Davis. IETF. September 2009. IETF Best Current Practice. URL: https://tools.ietf.org/html/bcp47

[bibtex]

BibTeX Format Description. URL: http://www.bibtex.org/Format/

[bidi]

Unicode Bidirectional Algorithm. Mark Davis; Aharon Lanin; Andrew Glass. Unicode Consortium. 4 February 2019. Unicode Standard Annex #9. URL: https://www.unicode.org/reports/tr9/tr9-41.html

[dcterms]

DCMI Metadata Terms. DCMI Usage Board. DCMI. 14 June 2012. DCMI Recommendation. URL: http://dublincore.org/documents/dcmi-terms/

[dom]

DOM Standard. Anne van Kesteren. WHATWG. Living Standard. URL: https://dom.spec.whatwg.org/

[dpub-aria-1.0]

Digital Publishing WAI-ARIA Module 1.0. Matt Garrish; Tzviya Siegman; Markus Gylling; Shane McCarron. W3C. 14 December 2017. W3C Recommendation. URL: https://www.w3.org/TR/dpub-aria-1.0/

[ecmascript]

ECMAScript Language Specification. Ecma International. URL: https://tc39.github.io/ecma262/

[fetch]

Fetch Standard. Anne van Kesteren. WHATWG. Living Standard. URL: https://fetch.spec.whatwg.org/

[html]

HTML Standard. Anne van Kesteren; Domenic Denicola; Ian Hickson; Philip Jägenstedt; Simon Pieters. WHATWG. Living Standard. URL: https://html.spec.whatwg.org/multipage/

[iana-link-relations]

Link Relations. URL: https://www.iana.org/assignments/link-relations/link-relations.xhtml

[iana-relations]

Link Relations. IANA. URL: https://www.iana.org/assignments/link-relations/

[iso8601]

Representation of dates and times. ISO 8601:2004.. International Organization for Standardization (ISO). 2004. ISO 8601:2004. URL: http://www.iso.org/iso/catalogue_detail?csnumber=40874

[json]

The application/json Media Type for JavaScript Object Notation (JSON). D. Crockford. IETF. July 2006. Informational. URL: https://tools.ietf.org/html/rfc4627

[json-ld]

JSON-LD 1.0. Manu Sporny; Gregg Kellogg; Markus Lanthaler. W3C. 16 January 2014. W3C Recommendation. URL: https://www.w3.org/TR/json-ld/

[onix]

ONIX for Books. URL: http://www.editeur.org/83/Overview

[publishing-linking]

Publishing and Linking on the Web. Ashok Malhotra; Larry Masinter; Jeni Tennison; Daniel Appelquist. W3C. 30 April 2013. W3C Note. URL: https://www.w3.org/TR/publishing-linking/

[PWP-UCR]

Web Publications Use Cases and Requirements. Joshua Pyle; Franco Alvarado. W3C. 19 February 2019. W3C Note. URL: https://www.w3.org/TR/pwp-ucr/

[rfc2046]

Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. N. Freed; N. Borenstein. IETF. November 1996. Draft Standard. URL: https://tools.ietf.org/html/rfc2046

[RFC2119]

Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[rfc5988]

Web Linking. M. Nottingham. IETF. October 2010. Proposed Standard. URL: https://tools.ietf.org/html/rfc5988

[RFC8174]

Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://tools.ietf.org/html/rfc8174

[rfc8288]

Web Linking. M. Nottingham. IETF. October 2017. Proposed Standard. URL: https://httpwg.org/specs/rfc8288.html

[schema.org]

Schema.org. URL: https://schema.org

[sri]

Subresource Integrity. Devdatta Akhawe; Frederik Braun; Francois Marier; Joel Weinberger. W3C. 23 June 2016. W3C Recommendation. URL: https://www.w3.org/TR/SRI/

[url]

URL Standard. Anne van Kesteren. WHATWG. Living Standard. URL: https://url.spec.whatwg.org/

[WCAG20]

Web Content Accessibility Guidelines (WCAG) 2.0. Ben Caldwell; Michael Cooper; Loretta Guarino Reid; Gregg Vanderheiden et al. W3C. 11 December 2008. W3C Recommendation. URL: https://www.w3.org/TR/WCAG20/

[WCAG21]

Web Content Accessibility Guidelines (WCAG) 2.1. Andrew Kirkpatrick; Joshue O Connor; Alastair Campbell; Michael Cooper. W3C. 5 June 2018. W3C Recommendation. URL: https://www.w3.org/TR/WCAG21/

[WebIDL]

Web IDL. Boris Zbarsky. W3C. 15 December 2016. W3C Editor's Draft. URL: https://heycam.github.io/webidl/

[webidl-1]

WebIDL Level 1. Cameron McCormack. W3C. 15 December 2016. W3C Recommendation. URL: https://www.w3.org/TR/2016/REC-WebIDL-1-20161215/

J.2 Informative references §

[ecma-404]

The JSON Data Interchange Format. Ecma International. 1 October 2013. Standard. URL: https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[link-relation]

Identifier: A Link Relation to Convey a Preferred URI for Referencing. H. Van de Sompel; M. Nelson; G. Bilder; J. Kunze; S. Warner. IETF. URL: https://tools.ietf.org/html/draft-vandesompel-identifier-00

[rfc3987]

Internationalized Resource Identifiers (IRIs). M. Duerst; M. Suignard. IETF. January 2005. Proposed Standard. URL: https://tools.ietf.org/html/rfc3987

[string-meta]

Requirements for Language and Direction Metadata in Data Formats. Addison Phillips; Richard Ishida. 2017-12-01. URL: https://w3c.github.io/string-meta/

[webschemas-a11y]

WebSchemas Accessibility. URL: http://www.w3.org/wiki/WebSchemas/Accessibility

↥