C2PA Technical Specification

PUBLIC DRAFT, 2021-08-29: Draft Specification (0.7)

Table of Contents

Chapter 1. Introduction

1.1. Overview

With the digital transformation of information sharing, the ability to trace the provenance of media has become critical. To address this issue at scale for publishers, creators and consumers, the Coalition for Content Provenance and Authenticity (C2PA) has developed this technical specification for establishing content provenance and authenticity. This specification has been, and continues to be, informed by scenarios, workflows and requirements gathered from industry experts and partner organizations, including the Project Origin Alliance and the Content Authenticity Initiative (CAI).

This specification is designed to enable global adoption of digital provenance techniques through the creation of a rich ecosystem of digital provenance enabled applications that can be utilized by creators, publishers, media consumers, regulatory bodies, and governmental agencies.

In developing this specification, the C2PA is focused on ensuring that it can be used in ways that respect privacy and personal control of data. In addition, this specification promotes tool availability for a wide range of organizations while meeting appropriate security requirements. Additionally, it is important that the specification does not negatively impact content accessibility for consumers.

1.2. Scope

This specification describes the technical aspects of the C2PA architecture; a model for storing and accessing cryptographically verifiable information whose trustworthiness can be assessed based on a defined trust model. Included in this document is information about how to create and process a C2PA manifest and its components, including the use of digital signature technology for enabling tamper-evidence as well as establishing trust.

Other documents from the C2PA will address specific implementation considerations such as expected user experiences and details of our threat and harms modelling.

1.3. Technical Overview

The C2PA information comprises a series of statements that cover areas such as asset creation, authorship, edit actions, capture device details, bindings to content and many other subjects. These statements, called Assertions, make up the provenance of a given asset and represent a series of trust signals that can be used by a human to improve their view of trustworthiness concerning the asset. Assertions are wrapped up with additional information into a digitally signed entity called a Claim.

The W3C Verifiable Credentials of individual actors that are involved in the creation of the assertions can be added to the C2PA information to provide additional trust signals to the process of assessing trustworthiness of the asset.

These assertions, claims, credentials and signatures are all bound together into a verifiable unit called a Manifest by a

hardware or software component called a Claim Generator. The set of manifests, as stored in the asset's Manifest Store, represent its provenance data.

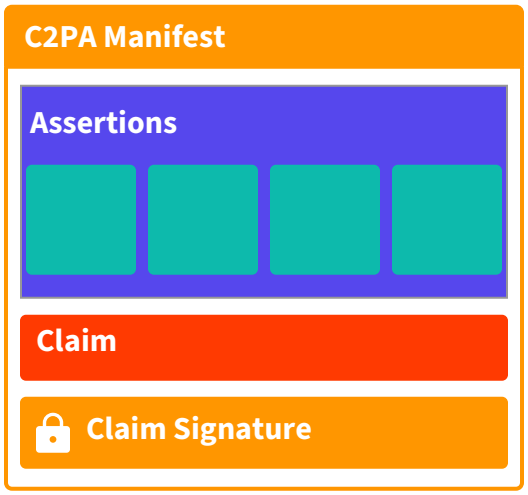


Figure 1. A C2PA Manifest and its constituent parts

1.4. Establishing Trust

The basis of making trust decisions in C2PA, our Trust Model, is the identity of the actor associated with the cryptographic signing key used to sign the claim in the active manifest. The identity of a signatory is not necessarily a human actor, and the identity presented may be a pseudonym, completely anonymous, or pertain to a service or trusted hardware device with its own identity, including an application running inside such a service or trusted hardware.

1.5. Example

A very common scenario will be a user (called an actor in the C2PA ecosystem) taking a photograph with their C2PA-enabled camera (or phone). In that instance, the camera would create a C2PA manifest containing some such assertions including information about the camera itself, a thumbnail of the image and some cryptographic hashes that bind the photograph to the manifest. These assertions would then be listed in the Claim, which would be digitally signed and then the entire manifest would be embedded into the output JPEG.

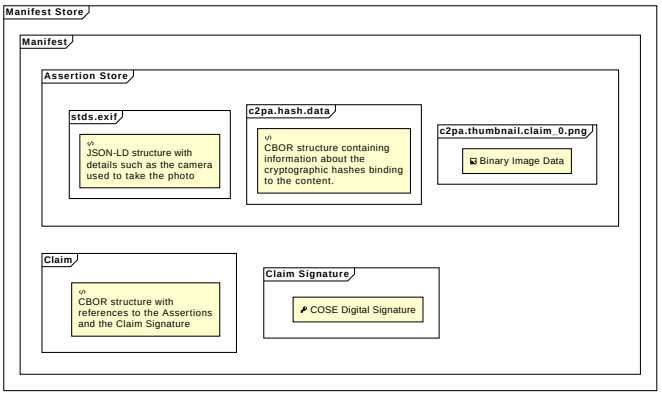


Figure 2. Photo manifest

A manifest consumer, such as a C2PA validator, could help users to establish the trustworthiness of the asset by first validating the digital signature and its associated credential. It can also check each of the assertions for validity and present the information contained in them, and the signature, to the user in a way that they can then make an informed decision about the trustworthiness of the digital content.

1.6. Design Goals

In the creation of the C2PA architecture, it was important to establish some clear goals for the work to ensure that the technology was usable across a wide spectrum of hardware and software implementations worldwide and accessible to all.

Some of those goals were:

- Maintain the provenance of the asset across multiple tools, from creation through all subsequent modification and publication/distribution.
- Support all standard asset formats supported by common authoring tools, across media types such as images, videos, audio, and documents.
- Create only the minimum required novel technology by relying on well-established techniques.
- Do not require cloud storage but allow for it.
- Allow flexibility in the nature of information stored.
- Allow for information to be subsequently redacted, provided that the author permits it.

Chapter 2. Glossary

2.1. Introductory terms

2.1.1. Actor

A human or non-human (hardware or software) that is participating in the C2PA ecosystem. For example: a camera (capture device), image editing software, cloud service or the person using such tools.

NOTE An organization or group of *actors* may also be considered an *actor* in the C2PA ecosystem.

2.1.2. Signer

An actor (human or non-human) whose credential's private key is used to sign the claim. The signer is identified by the subject of the credential.

2.1.3. Claim generator

The non-human (hardware or software) *actor* that generates the *claim* about an *asset* as well as the *claim signature*, thus leading to the *asset*'s associated *manifest*.

2.1.4. Manifest consumer

An *actor* who consumes an *asset* with an associated *manifest* for the purpose of obtaining the *provenance data* from the *manifest*.

2.1.5. Action

An operation performed by an *actor* on an *asset.* For example, "create", "embed", or "apply filter".

2.2. Assets and Content

2.2.1. Digital content

The portion of an *asset* that represents the actual content, such as the pixels of an image, along with any additional technical metadata required to understand the content (e.g., a colour profile or encoding parameters).

2.2.2. Asset metadata

The portion of an *asset* that represents non-technical information about the *asset* and its *digital content*, as may be stored via standards such as EXIF or XMP.

2.2.3. Asset

A file or stream of data containing *digital content*, *asset metadata* and optionally, a C2PA *manifest*.

NOTE	For the purposes of this definition, we will extend the typical definition of "file" to include cloud-native and dynamically generated data.

2.2.4. Derived asset

A derived asset is an *asset* that is created by starting from an existing *asset* and performing *actions* to it that modify its *digital content* and *asset metadata*.

EXAMPLE: An audio stream that has been shortened or a document where pages have been added.

2.2.5. Asset rendition

A representation of an *asset* (either as a part of an *asset* or a completely new *asset*) where the *digital content* has had a 'non-editorial transformation' *action* (e.g., re-encoding or scaling) applied but where the *asset metadata* has not been modified.

NOTE	This is also referred to as a *Facsimile Asset*

EXAMPLE: A video file that is re-encoded for reduced screen resolution or network bandwidth.

2.2.6. Composed asset

A composed asset is an *asset* that is created by building up a collection of multiple parts or fragments of *digital content* (referred to as ingredients) from one or more other *assets*. When starting from an existing *asset*, it is a special case of a *derived asset* - however a *composed asset* can also be one that starts from a "blank slate".

EXAMPLE: A video created by importing existing video clips and audio segments into a "blank slate". An image where another image is imported and super-imposed on top of the starting image.

2.3. Core Aspects of C2PA

2.3.1. Assertion

A data structure which represents a statement asserted by an *actor* concerning the *asset*. This data is a part of the *manifest*.

2.3.2. Claim

A digitally signed and tamper-evident data structure that references a set of *assertions* by one or more *actors*, concerning an *asset*, and the information necessary to represent the *content binding*. If any *assertions* were redacted,

then a declaration to that effect is included. This data is a part of the *manifest*.

2.3.3. Claim signature

The digital signature on the *claim* using the private key of an *actor*. The claim signature is a part of the *manifest*.

2.3.4. Manifest

The set of information about the *provenance* of an *asset* based on the combination of one or more *assertions* (including *content bindings*), a single *claim*, and a *claim signature*. A manifest can either be embedded into an *asset* or be external to its *asset*.

NOTE A manifest can reference other manifests.

2.3.5. Origin

The *manifest* in the *provenance data* which represents the method or device that initially created the *asset*.

NOTE Details on how one determines which *manifest* is the *origin* are left for specification.

2.3.6. Active Manifest

The last manifest in the list of manifests which is the one with the set of *content bindings* that are able to be validated.

2.3.7. Provenance

The logical concept of understanding the history of an *asset* and its interaction with *actors* and other *assets*, as represented by the *provenance data*.

2.3.8. Provenance data

The set of *manifests* for an *asset* and, in the case of a *composed asset*, its ingredients.

2.3.9. Content binding

Information that associates *digital content* to a specific *manifest* associated with a specific *asset*, either as a *hard binding* or a *soft binding*.

2.3.10. Hard binding

One or more cryptographic hashes that uniquely identifies either the entire *asset* or a portion thereof.

2.3.11. Soft binding

A content identifier that is either (a) not statistically unique, such as a *fingerprint*, or (b) embedded as a *watermark* in the identified content.

2.3.12. Trust signals

The collection of information that can inform an *actor's* judgment of the trustworthiness of an *asset*. These are in addition to the *signer* of a *claim*, upon which the fundamental trust model relies.

2.4. Additional Terms

2.4.1. Fingerprint

A set of inherent properties computable from *digital content* that identifies the content or near duplicates of it.

EXAMPLE: An *asset* can become separated from its *manifest* due to removal or corruption of *asset* metadata. A *fingerprint* of the *digital content* of the *asset* could be used to search a database to recover the *asset* with an intact *manifest*.

2.4.2. Watermark

Information incorporated into the *digital content* (perceptibly or imperceptibly) of an *asset* which can be used, for example, to uniquely identify the *asset* or to store a reference to a *manifest*.

2.4.3. Provenance datastore

A repository into which C2PA *manifests* can be placed, and which can be searched using a *content binding*.

2.5. Overview

This image shows how all these various elements come together to represent the C2PA architecture.

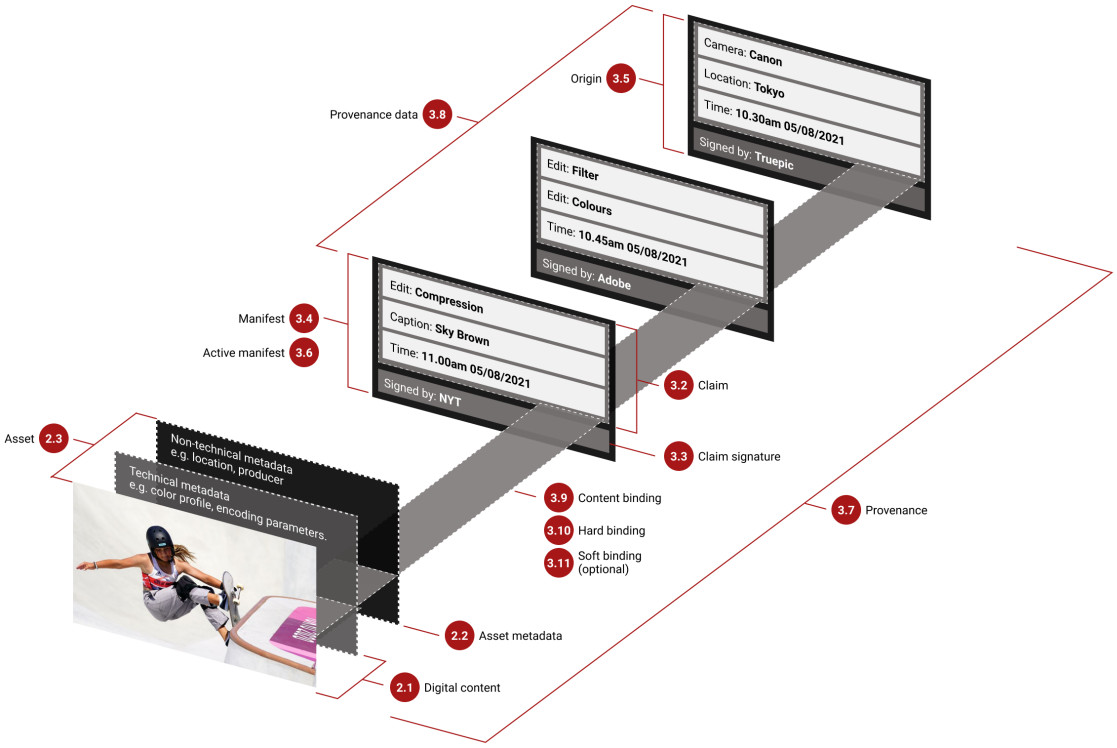


Figure 3. Elements of C2PA

Chapter 3. Normative References

3.1. Core Formats

- CBOR
- JSON
- JSON-LD
- JPEG universal metadata box format (JUMBF)
- ISO Base Media File Format (BMFF)

3.2. Schemas

- CDDL
- JSON Schema
- Dublin Core Metadata Initiative

3.3. Digital & Electronic Signatures

- X.509 Certificates
- JSON Web Algorithms (JWA)
- CBOR Object Signing and Encryption (COSE)
- Using RSA Algorithms with COSE Messages
- Online Certificate Status Protocol (OCSP)
- Internet X.509 PKI Time-Stamp Protocol
- CBOR Object Signing and Encryption (COSE): Header parameters for carrying and referencing X.509 certificates
- Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
- Internet X.509 Public Key Infrastructure: Additional Algorithms and Identifiers for DSA and ECDSA
- PKCS #1: RSA Cryptography Specifications Version 2.2
- JSON Advanced Electronic Signatures (JAdES)
- US Secure Hash Algorithms

3.4. Other

- eXtensible Metadata Platform (XMP)
- JSON-LD serialization of XMP
- IPTC Photo Metadata Standard
- EXIF
- UUID

Chapter 4. Standard Terms

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119, and RFC 8174 when they appear in any casing (upper, lower or mixed).

Chapter 5. Assertions

5.1. General

It is expected that each of the actors in the system that creates or processes an asset will produce one or more **assertions** about when, where, and how the asset was originated or transformed. An assertion is labelled data, typically (though not required to be) in a CBOR-based structure which represents a declaration made by an actor about an asset. Some of these actors will be human and add human-generated information (e.g., copyright) while other actors are machines (software/hardware) providing the information they generated (e.g., camera type).

Some examples of assertions are:

- Exif information (e.g. camera information such as maker, lens)
- Actions performed on the asset (e.g., clipping, color correction)
- Thumbnail of the asset or its ingredients
- Content bindings (e.g., cryptographic hashes)

Certain assertions may be redacted by subsequent claims (see Section 5.7, “Redaction of Assertions”), but they cannot be modified once made as part of a claim.

5.2. Labels

Each assertion has a **label** defined either by the C2PA specifications or an external entity.

Labels are string values organized into namespaces using a period (`.`) as a separator. The namespace component of the label can be an entity, or a reference to a well-established standard (see ABNF below). The most common labels will be defined by the C2PA and will begin with `c2pa.`. Entity-specific labels shall begin with the Internet domain name (without the top level domain (TLD), if that TLD is `.com`) for the entity (e.g., `adobe.`, `bbc.`), similar to how Java packages are defined. Well-established standards can use the "stds." prefix when describing their namespace. They are also versioned with a simple incrementing integer scheme (e.g., `c2pa.claim.date.v2`). If no version is provided, it is considered as `v1`. The list of publicly known labels can be found in Chapter 17, *C2PA Standard Assertions*.

```
namespaced-label = qualified-namespace label
qualified-namespace = entity / ( "stds."  std-name )
entity = 1*( DIGIT / ALPHA / "-" )
std-name = 1*( DIGIT / ALPHA / "-" )
label = 1*("." 1(ALPHA / "_" ) *( DIGIT / ALPHA / "_" ) )
```

The period-separated components of a label follow the variable naming convention (`[a-zA-Z_][a-zA-Z0-9_]*`) specified in the POSIX or C locale, with the restriction that the use of a repeated underscore character (`__`) is reserved for labeling multiple assertions of the same type.

5.3. Versioning

When an assertion's schema is changed, it should be done in a backwards-compatible manner. This means that new fields may be added and existing ones may be marked as deprecated (ie. can be read, but never written), but existing fields shall not be removed. The label would now consist of an incremented version number, for example moving from `c2pa.ingredient` to `c2pa.ingredient.v2`.

Deprecated fields for C2PA standard assertions shall be indicated in Chapter 17, *C2PA Standard Assertions*. Tools which enable actors to create assertions shall prevent the actor from inserting data into deprectated assertion fields.

In addition, there are situations where a non-backwards compatible change is required. In that case, instead of increasing the label's version number, the assertion shall be given a new label. For example, `c2pa.ingredient` could be changed to the fictional `c2pa.component`.

5.4. Multiple Instances

Multiple assertions of the same type can occur in the same manifest, but since assertions are referenced by claims via their label, the assertion labels must be unique. This is accomplished by adding a double-underscore and and a monotonically increasing index to the label. For example, if a manifest contains a single assertion of type `stds.schema-org.CreativeWork`, then the assertion label will be `stds.schema-org.CreativeWork`. If a manifest contains three assertions of this type, the labels will be `stds.schema-org.CreativeWork`, `stds.schema-org.CreativeWork__1` and `stds.schema-org.CreativeWork__2`.

When a label includes a version number, that version number is part of the label itself. As such, when there are multiple instances, the instance number continues to follow the label - e.g., `c2pa.ingredient.v2__2`.

5.5. Assertion Store

The set of assertions referenced by a claim in a manifest are collected together into a logical construct that is referred to as the **assertion store**. When embedded in an asset, the assertions shall be stored as described in Section 10.1, "Use of JUMBF", but when stored externally (e.g., in the cloud) they may be stored in any fashion.

For each manifest, there is a single assertion store associated with it. However, as an asset may have multiple manifests associated with it, each one representing a specific series of assertions, there may be multiple assertion stores associated with an asset.

5.6. Embedded vs Externally-Stored Data

As mentioned above, the entire assertion store (and all of the assertions that it contains) may be located externally from the asset (e.g., in the cloud). When the assertion store is embedded in an asset, it is still possible for an individual assertion to be located internally or externally. Additionally, just the data for a given assertion may also be located externally via a cloud data assertion (see Section 17.8, "Cloud Data").

5.7. Redaction of Assertions

Assertions that are present in an asset-embedded assertion store (regardless of whether their data is embedded or not) may be removed from future versions of the asset. This process is called redaction.

When an assertion is redacted, then either the redacted assertion is removed from the assertion store, or the labeled assertion container is retained but the assertion data itself is replaced with zeros.

Neither authoring tools nor claims generators shall redact assertions with a label of `c2pa.action` as this assertion type represents essential information in understanding the history of an asset. Additionally, such tools should take care not to redact assertions when they believe a human actor has a clear intention about its continued importance and propagation (e.g., `stds.schema-org.CreativeWork` assertions where the `copyright` fields have been filled out).

As part of the process of redacting an assertion, a record that something was removed is added to the claim. Because each assertion's URI reference includes the assertion label, it is also known what type of information (e.g., thumbnail, IPTC metadata, etc.) was removed. This enables both humans and machines to apply rules to determine if the removal was acceptable.

Chapter 6. Unique Identifiers

Every asset that is referenced from an assertion or claim shall be referenced via one or more unique identifiers. These identifiers are used in various parts of a C2PA-enabled workflow, such as when identifying it as an ingredient in a derived or composed asset.

6.1. Using XMP

When an asset contains embedded XMP, that XMP shall include (at least) values for `xmpMM:DocumentID` and `xmpMM:InstanceID` as defined in XMP Specification Part 2, 2.2. If an asset does not contain XMP at the time a claim is made, and the type of the asset supports it, an embedded XMP packet may be created as part of the process, and the identifiers shall be added to it.

NOTE

NOTE 1

Some asset types are not suited for embedded XMP (e.g., text). It is possible to create XMP as a sidecar.

NOTE

NOTE 2

The Adobe XMP Toolkit SDK can be used to create and modify XMP in various asset types. Alternatively, since XMP is serialized as XML+RDF, any standard XML library can be used for the purpose of working with XMP (though asset specific processing would be left up to the processor).

6.2. Other Identifiers

Instead of using XMP, a unique identifier for an asset could be a URI defined by standards such as EIDR and DOI.

Another standard unique identifier for an asset could be the cryptographic hash of the asset. When this method is used, the hash shall be represented using a standard RFC 4122 UUID following the recommendations at https://datatracker.ietf.org/doc/html/draft-thiemann-hash-urn-01 .

NOTE

EDITORS NOTE

Other methods may be defined here as they are developed.

6.3. URI References

Assertions and claims, whether they are stored internally to the asset (ie. embedded) or stored externally to the asset (e.g., in the cloud), shall be referenced via JUMBF URI references as defined in ISO 19566-5, C.2. This URI shall be used as part of a `hashed_uri` data structure:

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "$id": "http://ns.c2pa.org/hashed-uri/v1",
  "type": "object",
  "description": "The data structure used to store a reference to a URL and its hash",
  "definitions": {
    "JUMBF_URI": {
      "$id": "#JUMBF_URI",
      "description": "JUMBF URI reference",
      "type": "string",
      "anyOf": [
        {
          "pattern": "https:\/\/[-a-zA-Z0-9@:%._\\+~#=]{2,256}\\.[a-z]{2,6}\\b[-a-zA-Z0-
9@:%_\\+.~#?&//=]*"
        },
        {
          "pattern": "^self#jumbf=[\\w\\d][\\w\\d\\./:-]+[\\w\\d]$"
        }
      ]
    }
  },
  "examples": [
    {
      "url": "self#jumbf=c2pa/acme:urn:uuid:F9168C5E-CEB2-4faa-B6BF-
329BF39FA1E4/c2pa.assertions/c2pa.claim.date",
      "alg": "sha256",
      "hash": "hoOspQQ1lFTy/4Tp8Epx670E5QW5NwkNR+2b30KFXug="
    }
  ],
  "required": ["url", "hash"],
  "properties": {
    "url": {
      "$ref": "#/definitions/JUMBF_URI",
      "description": "JUMBF URI reference"
    },
    "alg": {
      "type": "string",
      "minLength": 1,
      "description": "A string identifying the cryptographic hash algorithm used to compute
all hashes in this claim, taken from the C2PA hash algorithm identifier list. If this field
is absent, the hash algorithm is taken from an enclosing structure as defined by that
structure. If both are present, the field in this structure is used. If no value is present
in any of these places, this structure is invalid; there is no default."
    },
    "hash": {
      "type": "string",
      "minLength": 1,
      "description": "CBOR byte string containing the hash value"
    }
  },
  "additionalProperties": false
}
```

NOTE This syntax enables these various objects to live either embedded or stored in the cloud while also providing the important hash value for validation purposes.

The hash is performed over the canonical serialization of the assertion data, omitting the surrounding container. For example, if the assertion type is `c2pa.actions`, then the hash is calculated over the canonical serialization

serialization of `actions-map`.

If a URI reference is an external URL (as defined in ISO 19566-5, C.2), then the domain of that URL shall be the same domain/organization associated with the signing certificate used on the claim signature. Also, for URI references to assertions stored externally to the asset, the label of the assertion shall be included in the query parameter of the URL, such as `http://c2pa.adobe.com/assertions/123456?cai=stds.exif`.

Chapter 7. W3C Verifiable Credentials

7.1. General

In some use cases, the actors in the system may wish to provide their own W3C Verifiable Credential, as they exist at that moment in time, to the claim generator to have them associated with one or more assertions. These actors may be individuals, groups or organizations.

W3C Verifiable Credentials are used in this specification to *decorate* the actors identified in assertions with more information, potentially providing additional trust signals. Although these W3C Verifiable Credentials can include proofs of their own authenticity, they are **not** a mechanism for verifying that a particular actor authorised a claim, assertion or piece of metadata. Any validation or usage of the W3C Verifiable Credential is out of scope of this specification and has no bearing on the C2PA Trust Model.

For example, conveying a W3C Verifiable Credential for the actor identified as the `author` in an assertion might link that author's ID with an email address, social media ID, or real name, or it might identify that actor as a member of a particular professional body, or provide other qualifications relevant to the actor's involvement in the asset.

Such credentials shall be compliant with the W3C Verifiable Credentials Data Model using the JSON-LD serialisation described there.

NOTE	JSON-LD serialization is mandated as it is the most commonly used of the three syntaxes presented in section 6 of the W3C Verifiable Credentials specification. It is also the one that aligns best with its extensibility model, which could be useful to some implementors.

An example of a compliant credential for an individual might be one issued by the National Press Photographers Association (NPPA), which links an identifier for a person to their name ("Bob Ross") and a statement about their membership of the NPPA. It might look like:

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "http://schema.org"
  ],
  "type": [
    "VerifiableCredential",
    "NPPACredential"
  ],
  "issuer": "https://nppa.org/",
  "credentialSubject": {
    "id": "did:nppa:eb1bb9934d9896a374c384521410c7f14",
    "name": "Bob Ross",
    "memberOf": "https://nppa.org/"
  },
  "proof": {
    "type": "RsaSignature2018",
    "created": "2021-06-18T21:19:10Z",
    "proofPurpose": "assertionMethod",
    "verificationMethod":
"did:nppa:eb1bb9934d9896a374c384521410c7f14#_Qq0UL2Fq651Q0Fjd6TvnYE-faHiOpRlPVQcY_-tA4A",
    "jws": "eyJhbGciOiJQUzI1NiIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19
      DJBMvvFAIC00nSGB6Tn0XKbbF9XrsaJZREWvR2aONYTQQxnyXirtXnlewJMB
      Bn2h9hfcGZrvnC1b6PgWmukzFJ1IiH1dWgnDIS81BH-IxXnPkbuYDeySorc4
      QU9MJxdVkY5EL4HYbcIfwKj6X4LBQ2_ZHZIu1jdqLcRZqHcsDF5KKylKc1TH
      n5VRWy5WhYg_gBnyWny8E6Qkrze53MR7OuAmmNJ1m1nN8SxDrG6a08L78J0-
      Fbas5OjAQz3c17GY8mVuDPOBIOVjMEghBlgl3nOi1ysxbRGhHLEK4s0KKbeR
      ogZdgt1DkQxDFxxn41QWDw_mmMCjs9qxg0zcZzqEJw"
  }
}
```


A W3C Verifiable Credential used with C2PA shall contain only a single `credentialSubject` and that `credentialSubject` shall have an `id` value.

NOTE

Although the example above and many examples in the W3C Verifiable Credentials data model specification use Decentralized Identifiers (DIDs) as the value of the `id` field, DIDs are not necessary for W3C Verifiable Credentials to be useful. Specifically, W3C Verifiable Credentials do not depend on DIDs and DIDs do not depend on W3C Verifiable Credentials. DID-based URLs are just one way to express identifiers associated with subjects, issuers, holders, credential status lists, cryptographic keys, and other machine-readable information associated with a W3C Verifiable Credential.

7.2. VCStore

The set of credentials in a manifest are collected together into a logical construct that is referred to as the **credential store** or **VCStore** (for short) and it shall be stored as described in Section 10.1, “Use of JUMBF”. Unlike the assertion store, the VCStore shall always be included in the JUMBF - it shall not be stored separately.

For each manifest, there is a VCStore associated with it. However, as an asset may have multiple manifests associated with it, there may be multiple VCStores associated with an asset.

7.3. Using Credentials

The Creative Work assertion may contain references to Persons or Organisations which are responsible for various roles and responsibilities to the Work. These references may contain W3C Verifiable Credentials via the `credential` field whose value is the hashed JUMBF URI to the specific credential in the VCStore. When present in the Person or Organization object, its `id` field shall match the `id` field present in the credentialSubject field of the VCs.

```
{
  "@context": "http://schema.org/",
  "@type": "CreativeWork",
  "copyrightHolder": {
    "name": "BBC",
    "legalName": "British Broadcasting Corporation",
    "id": "https://www.bbc.co.uk/",
    "credential": [
      {
        "url": "self#jumbf=c2pa/acme:urn:uuid:F9168C5E-CEB2-4faa-B6BF-
329BF39FA1E4/c2pa.credentials/https://www.bbc.co.uk/",
        "alg": "sha256",
        "hash": "Auxjtmax46cC2N3Y9aFmBO9Jfay8LEwJWzBUtZ0sUM8gA"
      }
    ]
  },
  "copyrightYear": 2021,
  "copyrightNotice": "Copyright © 2021 BBC."
}
```

7.4. Credential Security Considerations

In most W3C Verifiable Credential workflows, the information about the subject (e.g., the cryptographic keys) is fetched on demand at the time of validation. While that is an acceptable model, it does open up a possible attack vector by providing an attacker with an externally-visible signal about what the validator is validating. Therefore, C2PA also supports having the information captured and embedded at the time of signature. This not only prevents leakage, but also makes it very clear what data the signer is asserting about the credential's subject.

Chapter 8. Binding to Content

8.1. Overview

A key aspect to the C2PA manifest is the presence of one or more data structures that can uniquely identify portions of the asset. There are two types of bindings that are supported by C2PA - hard bindings and soft bindings. A hard binding (also known as a cryptographic binding) enables the verifier to ensure that (a) this manifest belong with this asset and (b) that the asset has not been modified, by determining values that can match only this asset and no other, not even other assets derived from it or renditions produced from it. A soft binding is computed from the digital content of an asset, rather than its raw bits. A soft binding is useful for identifying derived assets and asset renditions.

8.2. Hard Bindings

8.2.1. Hashing using byte ranges

The simplest type of hard binding that can be used to detect tampering is a cryptographic hashing algorithm, as described in Section 12.1, “Hashing”, over some or all of the bytes of an asset. This approach can be used on any type of asset.

When using this form of hard binding, one or more data hash assertions is used to define the range of bytes that are hashed (and those that are not). Because each data hash assertion defines a byte range and optional URL, it is flexible enough to be usable whether the asset is a single binary or represented in multiple chunks or portions, local or remote.

8.2.2. Hashing a BMFF-formatted asset

If the asset is based on ISO BMFF then a hard binding optimized for the box-based format (called BMFF-based hash assertions) may be used instead.

For a monolithic mp4 file asset where the `mdat` box is verified as a unit, the assertion is verified nearly identically to a data hash assertion. It simply uses a box exclusion list instead of byte ranges to define the range of bytes that are hashed (and those that are not).

For a monolithic mp4 file asset where the `mdat` box is verified piecemeal or an asset composed of fragmented mp4 files, the assertion itself must be combined with chunk-specific hashing information which is located as specified in Section 10.2.2, “Embedding manifests into BMFF-based assets”. Verifying a given chunk requires first verifying the `merkle` field’s `initHash` over the corresponding initialization segment and then locating the correct entry in the `merkle` field’s `hashes` array and verifying it against the hash of the chunk’s data plus (if needed) deriving the hash using the other `hashes` specified in the chunk’s `c2pa` box.

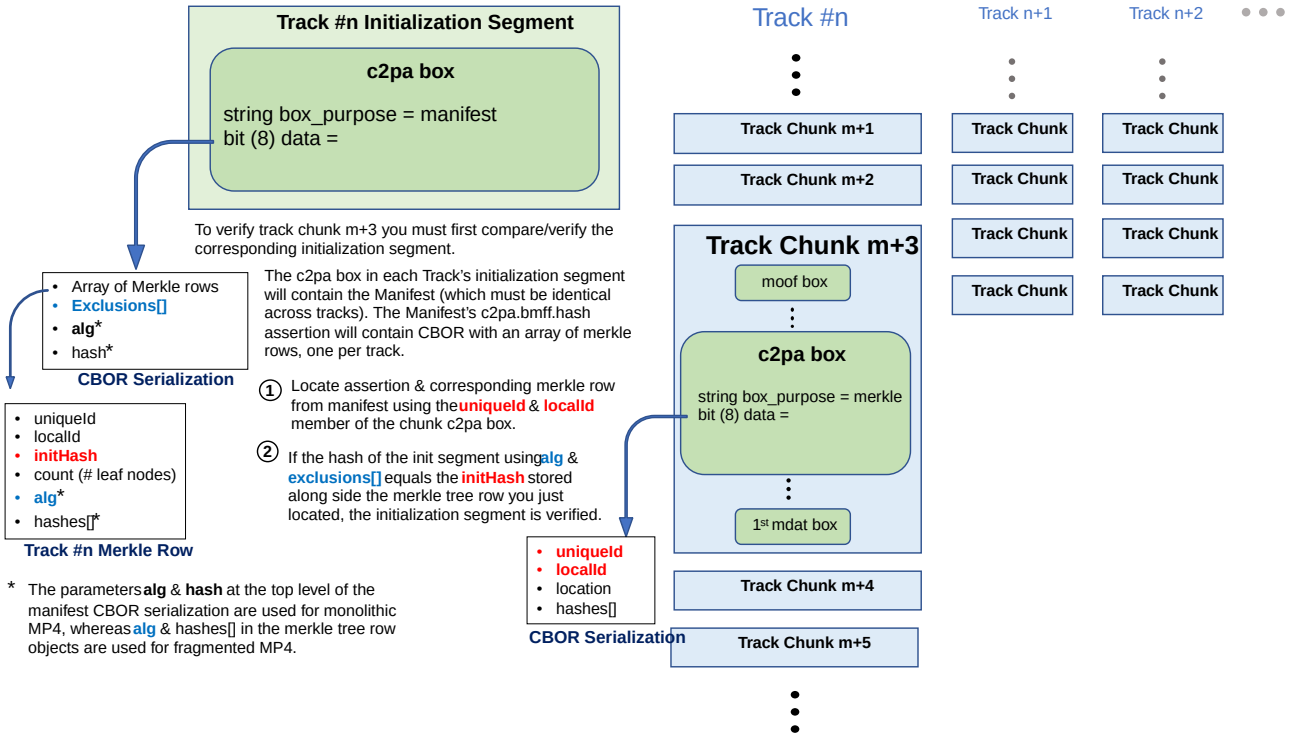


Figure 4. Verifying the initialization segment

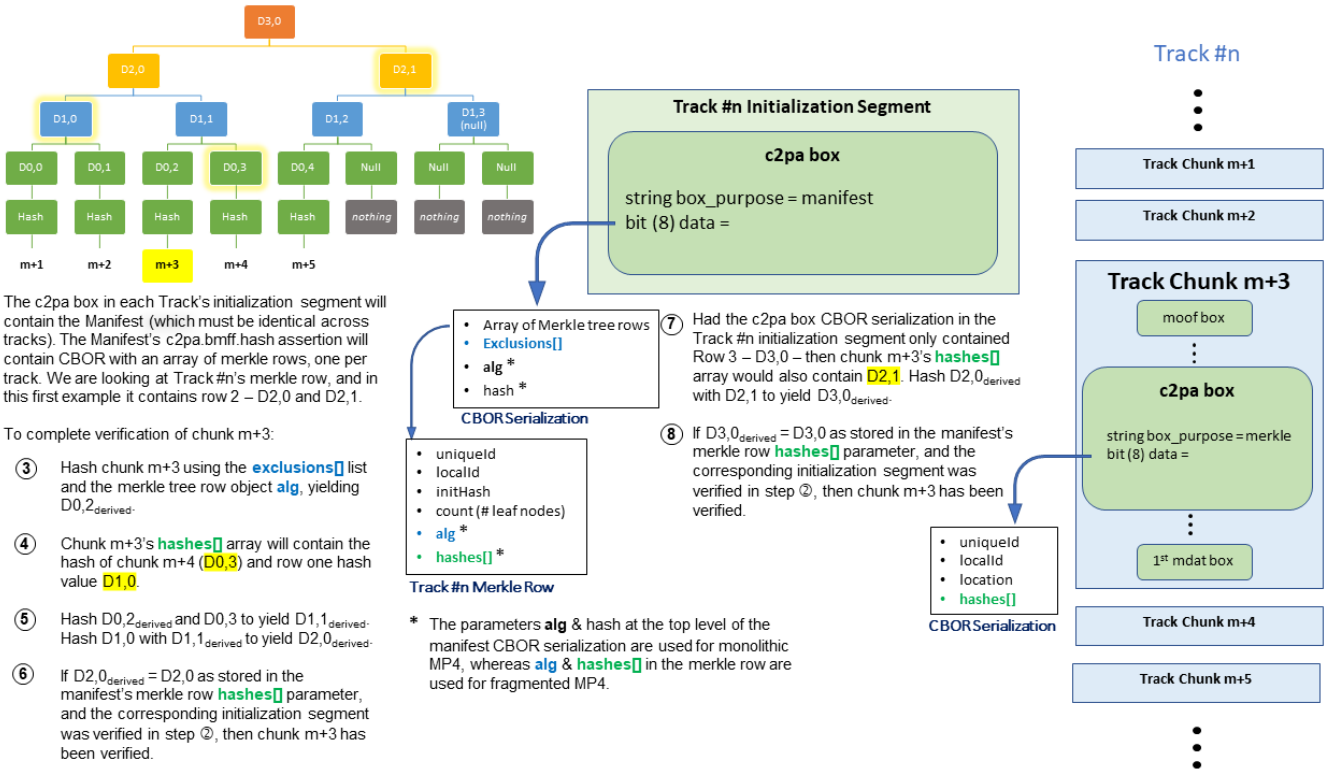


Figure 5. Verifying the chunk's data

8.2.3. Asset Metadata Bindings

In those workflows which embed XMP or other forms of asset metadata into the asset, the asset's asset metadata

should not be excluded by data hash assertions.

This means that by default all asset metadata (including Exif metadata and IPTC metadata in either IPTC-IIM or XMP format) will be included in the data hash assertions, but with no provenance information such as **who** made the claims.

To explicitly assert the same claims in a C2PA assertion with verifiable provenance, the Exif or IPTC fields should be copied to a `stds.exif` or `stds.iptc.photo-metadata` assertion, as appropriate (see Section 17.13, "Exif Information" and Section 17.14, "IPTC Photo Metadata").

NOTE We recommend that existing Exif, IPTC-IIM and/or XMP asset metadata be left untouched in the asset. This will allow for compatibility with tools which do not yet support C2PA metadata.

8.3. Soft Bindings

Soft bindings are described using soft binding assertions such as via a perceptual hash computed from the digital content or a watermark embedded within the digital content. These soft bindings enable digital content to be matched even if the underlying bits differ, for example due to an asset rendition in a different resolution or encoding format. Additionally, should a C2PA manifest be removed from an asset, but a copy of that manifest remains in a provenance store elsewhere, the manifest and asset may be matched using available soft bindings.

Because they serve a different purpose, a soft binding shall not be used as a *hard binding*.

All soft bindings shall be generated using one of the algorithms listed as supported by the C2PA soft binding algorithm registry. This registry will define:

- A list of algorithms that are allowed for generating soft bindings of new content as well as required for validating or locating existing content (the "allowed" list), and
- A list of algorithms that are required to be supported for validating or locating existing content but are not allowed for generating soft bindings of new content (the "deprecated" list).

This list will establish the allowed algorithms for creating a soft binding from the digital content of an asset (or part thereof) and a string algorithm identifier to be used as the algorithm identifier in the corresponding field.

Different versions of an algorithm must be defined using different string algorithm identifiers.

An algorithm when described in the registry must define the content types over which it is applicable.

An algorithm when described in the registry should reference technical documentation sufficient for the soft binding algorithm to be implemented.

An algorithm when described in the registry should be defined along with the names and values of all parameters affecting the operation of that algorithm. When doing so, it shall describe the manner in which those parameters must be encoded within the `alg-params` field of the soft binding assertion. An algorithm that is instantiated over a different parameter set will be considered a different algorithm.

An algorithm when described in the registry may also define an encoding scheme for specifying the portion of digital content over which a soft binding is computed (namely, the `extent` field of the `SOFTBINDING_SCOPE` structure within the soft binding assertion. An algorithm that encodes the `extent` differently will be considered a different algorithm.

It is recommended that the string identifiers for soft binding algorithms conform to how they are referred to in common practice.

There are no soft binding algorithms recommended in this version of the specification. If the C2PA's soft binding hash algorithm registry has not yet been established, then the registry (the "bootstrap registry") shall contain an empty deprecated list and an empty allowed list.

NOTE

The C2PA is currently evaluating various soft binding algorithms. One of the many possible options includes the ISCC - International Standard Content Code. The ISCC is an identifier and fingerprint for digital assets that supports all major content types (e.g., text, image, audio, video). The ISCC uses is similarity-preserving hashes generated both from metadata and content.

Chapter 9. Claims

9.1. Overview

A **claim** gathers together all the assertions about an asset from an actor at a given time including the set of assertions for binding to the content. The claim is then cryptographically hashed and signed as described in Section 9.3.2.4, “Signing a Claim”. A claim has all the same properties as an assertion including being assigned a label (`c2pa.claim`). It can also either be embedded into the asset or stored remotely (e.g., in the cloud).

9.2. Syntax

The CDDL Definition for this type is:

```
; CDDL schema for a claim map in C2PA
claim-map = {
  "claim_generator": ua-formatted-str-type, ; A User-Agent string formatted as per
http://tools.ietf.org/html/rfc7231#section-5.5.3, for including the name and version of the
claims generator that created the claim
  "signature": jumbf-uri-type, ; JUMBF URI reference to the signature of this claim
  "assertions": [1* $hashed-uri-map],
  "dc:format": tstr, ; media type of the asset
  ? "dc:title": tstr .size (1..max-tstr-length), ; name of the asset,
  ? "instanceID": tstr .size (1..max-tstr-length), ; uniquely identifies a specific version
of an asset
  ? "redacted_assertions": [1* jumbf-uri-type], ; List of hashed URI references to the
assertions of this claim
  ? "alg": tstr .size (1..max-tstr-length), ; A string identifying the cryptographic hash
algorithm used to compute all data hash assertions listed in this claim unless otherwise
overridden, taken from the C2PA data hash algorithm identifier registry. This provides the
value for the 'alg' field in data-hash and hashed-uri structures contained in this claim
  ? "alg_soft": tstr .size (1..max-tstr-length), ; A string identifying the algorithm used
to compute all soft binding assertions listed in this claim unless otherwise overridden,
taken from the C2PA soft binding algorithm identifier registry."
}

jumbf-uri-type = tstr .regexp "self#jumbf=[\\w\\d][\\w\\d\\.\/:-]+[\\w\\d]"

; TO DO, check the specific requirement of the claim generator string
ua-formatted-str-type = tstr
```

An example in CBOR-Diag is shown below:

```
{
  "claim_generator": "Joe's Photo Editor/2.0 (Windows 10)",
  "signature" : "self#jumbf=c2pa/acme:urn:uuid:F9168C5E-CEB2-4faa-B6BF-
329BF39FA1E4/c2pa.signature",
  "alg" : "sha256",
  "dc:format": "image/jpeg",
  "assertions" : [
    {
      "url": "self#jumbf=c2pa/acme:urn:uuid:F9168C5E-CEB2-4faa-B6BF-
329BF39FA1E4/c2pa.assertions/c2pa.hash.data",
      "hash": b64'U9Gyz05tmpftkoEYP6XYNsMnUbnS/KcktAg2vv7n1n8='
    },
    {
      "url": "self#jumbf=c2pa/acme:urn:uuid:F9168C5E-CEB2-4faa-B6BF-
329BF39FA1E4/c2pa.assertions/c2pa.thumbnail.claim.jpeg",
      "hash": b64'G5hfJwYeWTlflxOhmfCO9xDAK52aKQ+YbKNhRZeq92c='
    },
    {
      "url": "self#jumbf=c2pa/acme:urn:uuid:F9168C5E-CEB2-4faa-B6BF-
329BF39FA1E4/c2pa.assertions/c2pa.claim.ingredient",
      "hash": b64'Yzag4o5jO4xPyfANVtw7ETlbFSWZNfeM78qbSi8Abkk='
    }
  ],
  "redacted_assertions" : [
    "self#jumbf=c2pa/urn:uuid:5E7B01FC-4932-4BAB-AB32-
D4F12A8AA322/c2pa.assertions/stds.exif"
  ]
}
```

The Media Type of the ingredient shall be declared in `dc:format`. If present, the value of `dc:title` shall be a human-readable name for the asset.

If the asset contains XMP, then the asset's `xmpMM:InstanceID` should be used as the `instanceID`. When no XMP is available, then some other unique identifier for the asset shall be used as the value for `instanceID`.

NOTE Some field names, such as `dc:format` and `instanceID` have namespace prefixes as their names definitions are taken directly from the XMP standard. However, their usage in C2PA does not require the use of XMP.

The value of `claim_generator` is a human-readable string that will let a user know what software/hardware/system produced this Claim. This field shall be present and its value shall be a string that conforms to the User-Agent string format specified in section 5.5.3 of HTTP/1.1 Semantics and Content.

The `signature` field shall be present containing a URI reference to a claim signature.

The `assertions` field shall be present containing one or more URI references to the assertions being made by this claim. At least one of these assertions must be a hard binding assertion.

When present, the `redacted_assertions` field shall contain one or more URI references to redacted assertions.

9.3. Creating a Claim

9.3.1. Creating Assertions

Before the claim can be finalized, all assertions must be created and stored in a newly created C2PA Assertion Store. This includes all of the content binding assertions that need to be present to ensure the asset is tamper-evident. A claim shall include one or more content binding assertions in its list of assertions and the asset's digital content (e.g., pixels in an image) shall not be excluded by any of the content binding assertions. It may still not be possible to know all of the binding information at the time of claim creation, in which case use the multiple step processing method to setup and then later fill-in the information.

9.3.2. Preparing the Claim

9.3.2.1. Adding Assertions and Redactions

The claim shall contain the `assertions` field and its value is a list of all of the URI references for all assertions that were added to the assertion store that are being "claimed" by this claim. At least one of the assertions shall be either a data hash assertion or a BMFF-based hash assertion.

If any assertions in ingredient claims are being redacted, their URI references shall be added to list which is the value of the `redacted_assertions` field.

9.3.2.2. Adding Ingredients

In many authoring scenarios, an actor does not create an entirely new asset but instead brings in other existing assets on which to create their work - either as a derived asset, a composed asset or an asset rendition. These existing assets are called ingredients and their use is documented in the provenance data through the use of an ingredient assertion. When an ingredient contains one or more C2PA manifests, those manifests must be inserted into this asset to ensure that the provenance data is kept intact. Such ingredient manifests are added to the JUMBF as described in Section 10.1.1, "C2PA Box details".

9.3.2.3. Connecting the Signature

The signature cannot be part of the signed payload, but since its label is pre-defined, then the full URI reference is also known. As such, we can include that in the claim by setting the value of the `signature` field of the claim to that URI reference.

NOTE	This provides the explicit binding of the claim to its signature.

9.3.2.4. Signing a Claim

Producing the signature is specified in Section 12.2, "Digital Signatures". The CBOR claim document as it appears in the C2PA Claim box is the payload, and the "detached mode" is used to produce a `COSE_Sign1` structure. The resulting `COSE_Sign1` structure is written into the C2PA Claim Signature box.

NOTE

A more typical structure would be to expand the claim schema to have a "to-be-signed" or "payload" member which contains what is currently the entire claim schema, and then add an additional field to contain the detached COSE signature. This is what X.509 does, and what COSE does in the default (non-detached) case where the payload is included encoded as a field of the `COSE_Sign1` structure. As there is no expectation CBOR is directly human-readable, the recommendation would be to use the `COSE_Sign1` structure with the payload included, as then there will be no need for C2PA to specify its own container format for the signature.

9.3.2.5. TimeStamps

If possible, the signer should use a Trust Service Provider (specifically a Trusted TimeStamp Provider) to generate a trusted token (time-mark or time-stamp token) proving that the signature itself actually existed at a certain date and time and incorporate that into the signature. All timestamps shall be done as described in RFC3161 as an unprotected header parameter named `sigTst` in the `COSE_Sign1` structure following the definition in JAdES section 5.3.4.

The `MessageImprint` for the `TimeStampReq` structure shall be computed by creating the `ToBeSigned` value in section 4.4 of RFC 8152, using "CounterSignature" as the context string for the `Sig_structure`. The `payload` for the `Sig_structure` is as described in Section 9.3.2.4, “Signing a Claim”. The `ToBeSigned` value is then hashed using an approved hash algorithm that the time stamping authority (TSA) supports, and that hash algorithm and value are placed in the `MessageImprint`. The `certReq` boolean of the `TimeStampReq` structure must be asserted in the request to the TSA, to ensure its certificate is provided in the response. The serialized content of the `TimeStampToken` received in reply is then stored as the `sigTst` header’s value.

The serialized content of the `TimeStampToken` received in reply shall then be stored as the `val` property of a `tstToken` element of the `tstTokens` array property of the `tstTokens` element of the `sigTst` header as defined by JAdES section 5.3.4 and its JSON schema, except with the modification that the content of `val` is a byte string containing the content of the `TimeStampToken`, and not a Base64-encoded version of the same. The `tstTokens` array shall contain at least one element.

NOTE

Recall that an "unprotected" header only means it is not included in the protected payload of the claim. A timestamp, like all countersignatures, is itself signed (and so brings its own integrity protection) and has to be produced after the claim is signed, and so cannot be a part of the claim’s protected payload.

9.3.2.6. Credential Revocation Information

If the signer’s credential type supports querying online credential status, and the credential contains a pointer to a service to provide timestamped credential status information, the signer should query the service, capture the response, and store it in the manner described for the signer’s credential type in the Trust Model. If credential revocation information is attached in this manner, a trusted timestamp must also be obtained after signing, as described in Section 9.3.2.5, “TimeStamps”.

9.3.3. Referencing the Active Manifest

In order to ensure that a validator can quickly determine the presence of a C2PA manifest, it may be useful to embed information in the asset that is easily accessed in a non-format specific manner.

When the asset has embedded XMP, the method used shall be the existence of a **dcterms:provenance** key in the XMP with a value that is the URI reference to the active C2PA Manifest. It would look something like:

```
<dcterms:provenance>self#jumbf=c2pa/acme:urn:uuid:F9168C5E-CEB2-4faa-B6BF-
329BF39FA1E4</dcterms:provenance>
```

NOTE

EDITORS NOTE

Other non-XMP-based methods will be defined here as they are developed.

9.3.4. Examples

9.3.4.1. Single Claim

Here is a visual representation of an image containing a single claim with multiple assertions that have been embedded inside it.

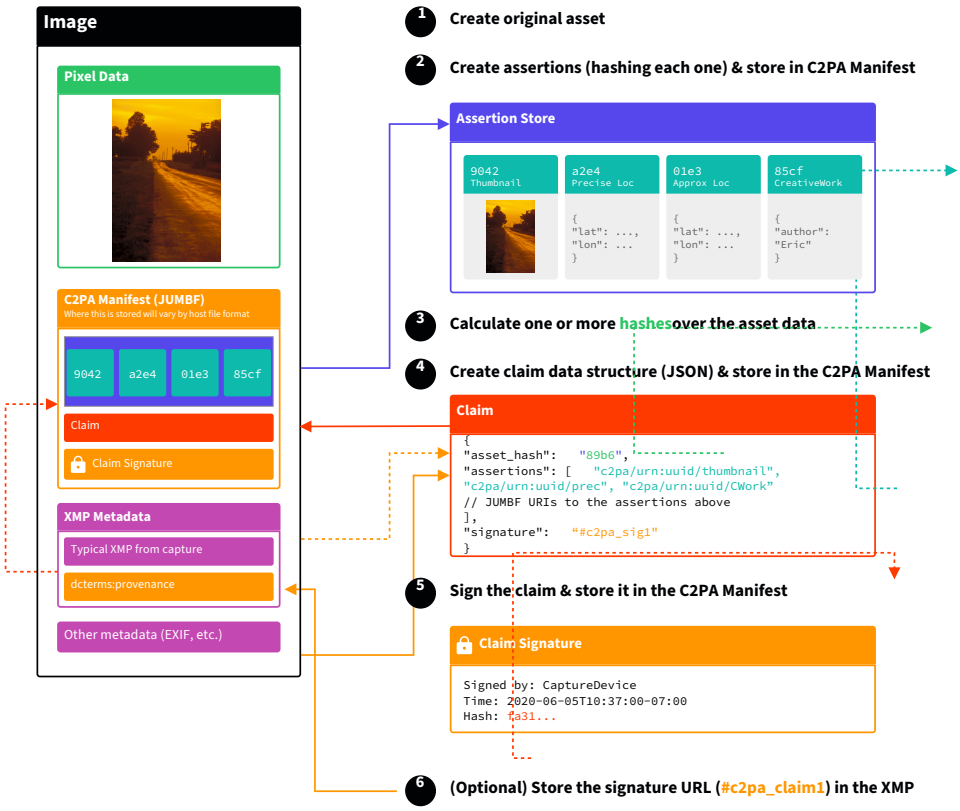


Figure 6. A single claim with assertions

9.3.4.2. Multiple Claims

In this example of creating a second claim for the previous example, one of the original assertions has been redacted from the previous claim. The visual representation for this scenario would look like:

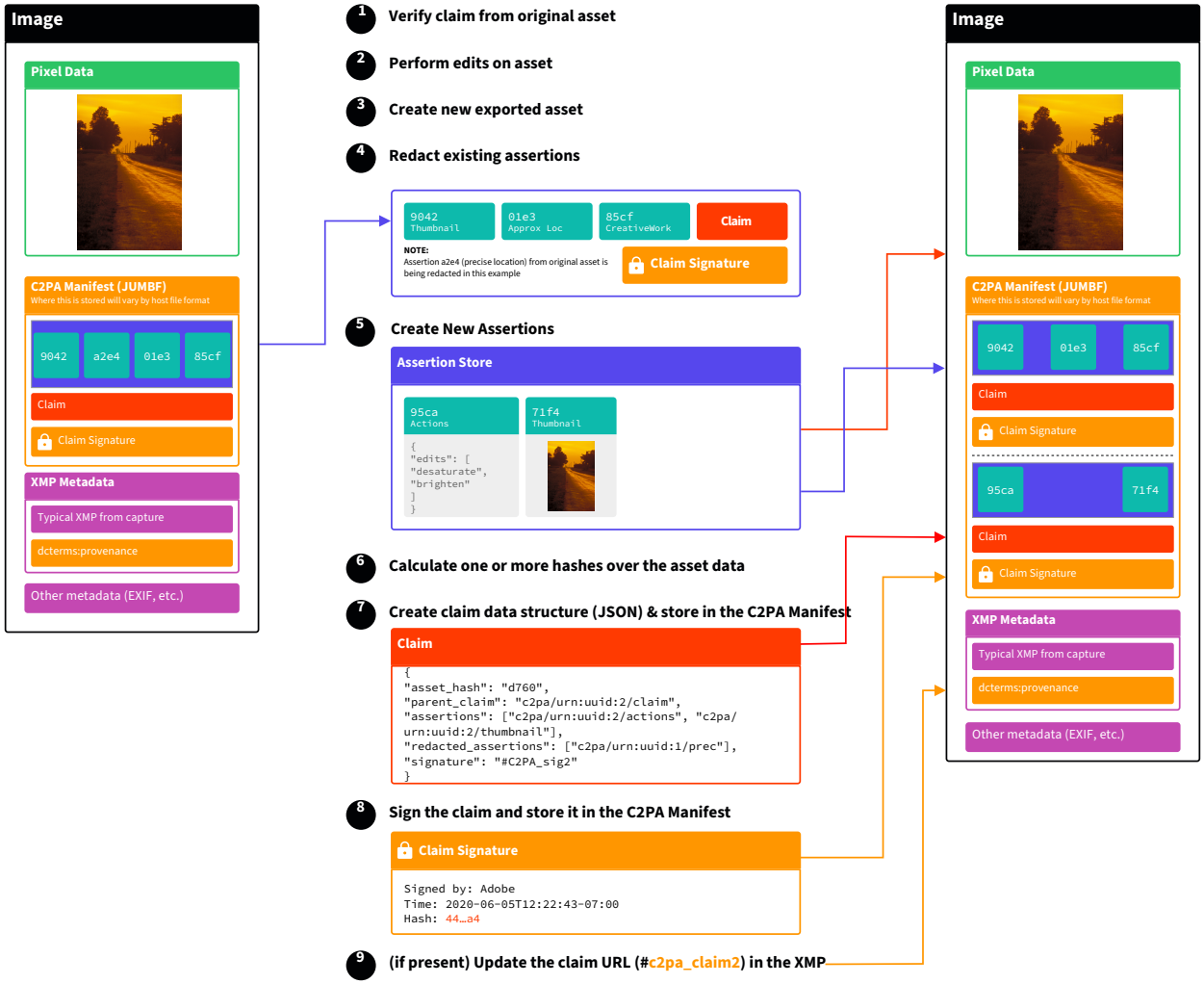


Figure 7. Redacting assertions in a secondary claim

9.4. Multiple Step Processing

There are situations where it is technically necessary to create & embed the manifest into an asset before the actual data of the asset has been fully created. For example, in JPEG-1 files, the entire C2PA Manifest Store must appear in the file before the image data - but if the manifest is to include cryptographic hashes of the image data, in the form of data hash assertions, how to achieve this?

To accomplish this, a multiple step approach is taken, similar to how signatures in PDF are done.

9.4.1. Prepare the XMP

For those C2PA-enabled assets that contain embedded XMP, start by creating the XMP data stream including adding

the active manifest reference and then serialize it into the asset in the standard location reserved for it in the format of the asset.

NOTE	While it is possible to add the XMP data to the list of exclusions in a data hash assertion, doing so is not recommended as it would remove tamper detection of that asset metadata.

9.4.2. Create content bindings

As mentioned in Section 9.3.1, “Creating Assertions”, a claim shall include one or more content binding assertions in its list of assertions.

In many cases, such as with JPEG-1, it is not possible to hash the asset in its entirety because the manifest will be embedded in the middle of the file, so the size or location manifest data will not be known at the time the asset hash is calculated. This circular dependency is avoided by allowing exclusion ranges to be specified during hashing. When exclusion ranges are specified, a single hash is performed, but only over the asset ranges that are not in any of the exclusions.

For example, if a manifest is embedded in the center of a JPEG-1 file in an APP11 segment, then the claim creator can exclude the APP11 segment(s) from the hash calculation.

In order to prevent insertion attacks, it is desirable to have only a single exclusion range when possible. When the size or location (or both) of the manifest in the asset is not known, then the `start` and `length` values in the data hash assertion shall both be zero and the size of the `pad` value should be large enough to accommodate writing in the values in the second pass. At least 16 bytes is recommended.

Create the data hash assertion and add it to the assertion store.

9.4.3. Create a temporary Claim & Signature

Add the newly created data hash assertion reference to the claim’s assertion list providing a temporary hash value, such as empty spaces.

At this point, the temporary claim is complete and can be added to the C2PA Manifest being created.

Since the claim is only temporary at this time, it is not possible to sign it, so an empty Claim Signature box should also be written that is large enough to include the final data (once produced). It is recommended to use `0` bytes to fill the signature.

9.4.4. Complete the C2PA Manifest

At this point all of the boxes that comprise the entire C2PA Manifest for the asset are completed and can be (if not already) constructed into its final form. The asset’s C2PA Manifest, along with the manifests of any ingredients, are combined together to form the complete C2PA Manifest Store. The C2PA Manifest Store can then be embedded into the asset as discussed in Section 10.2.2, “Embedding manifests into BMFF-based assets”.

9.4.5. Going back and filling in

Now that the C2PA Manifest Store has been embedded into the asset, the starting offset and the length of the active manifest can be updated in its data hash assertion. It is necessary that when doing so, you don't change the size of the assertion's box, only its data. This is done by adjusting the value of the `pad` field to be the necessary length to "fill up" the remaining bytes.

Once the data hash assertion has been updated, it can be hashed and the hash written over the empty spaces that were used previously to hold the location.

The claim is now complete, and it can be hashed and signed as described in Section 9.3.2.4, "Signing a Claim", with the resultant signature filling the pre-allocated space. The extra `0` bytes can either be left alone or a new `free` JUMBF box can be added to mark the space as such.

Chapter 10. Manifests

10.1. Use of JUMBF

In order to support many of the requirements of C2PA, C2PA Manifests needed to be stored (serialized) into a structured binary data store that enabled some specific functionality including:

- Ability to store multiple manifests (e.g., parents and ingredients) in a single container
- Ability to refer to individual elements (both within and across manifests) via URIs
- Ability to clearly identify the parts of an element to be hashed
- Ability to store pre-defined data types used by C2PA (e.g., JSON and CBOR)
- Ability to store arbitrary data formats (e.g., XML, JPEG, etc.)

In addition to supporting all of the requirements above, our chosen container format - JUMBF, ISO 19566-5 - is also natively supported by the JPEG family of formats and is compatible with the box-based model (i.e. ISOBMFF, ISO 14496-12) used by many common image and video file formats. Using JUMBF enables all the same benefits (and a few extras, such as URI References) while being able to work with classic image formats, such as JPEG/JFIF and PNG as well as 3D and document (e.g., PDF) formats.

NOTE

Since most of the standard assertions, as well the claim signature, are serialized as CBOR, using CBOR for the entire manifest was considered but not chosen because CBOR is not a container format. It could be used as one through having to re-define how CBOR would be used to provide the features natively supported by JUMBF.

For example, to store a "blob of JSON" inside of CBOR, and know that it is JSON (and not some other format) would require designing a data structure for storing such things. Then the parent structure would need to be defined as to how to carry that structure. This same concept would also have to be done for each of the native features of JUMBF.

While it would certainly be possible to re-implement all of the required functionality entirely in CBOR, it would be a lot of work and would not fully remove the need for a JUMBF/BMFF parser in all implementations.

10.1.1. C2PA Box details

When C2PA data is serialized into a JUMBF-compatible structure, all of its data is referred to as the C2PA Manifest Store. An example C2PA Manifest Store, with a single manifest, might look like this:

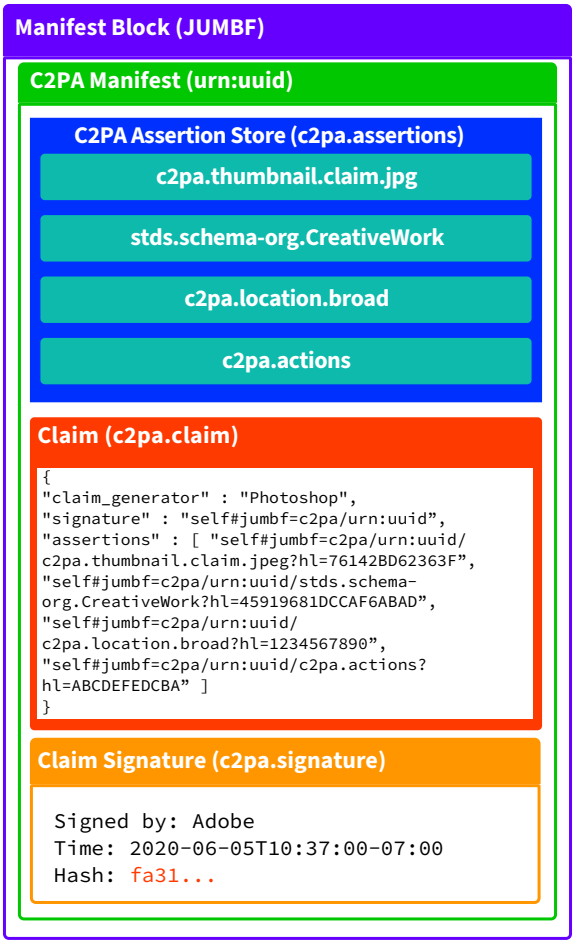


Figure 8. C2PA Manifest Store

The C2PA Manifest Store is a JUMBF superbox composed of a series of other JUMBF boxes and superboxes, each identified by their own UUID and label in their JUMBF Description box. The C2PA Manifest Store shall have a label of `c2pa`, a UUID of `0x63327061-0011-0010-8000-00AA00389B71` (`c2pa`) and shall contain one or more C2PA Manifest superboxes.

Each C2PA Manifest superbox shall contain the data created at the time a claim is issued including the C2PA Assertion Store, a C2PA Claim, and a C2PA Claim Signature. It may also contain a C2PA Credential Store. The UUID for each C2PA Manifest shall be `0x63326D61-0011-0010-8000-00AA00389B71` (`c2ma`), but in order to enable uniquely identifying each manifest, they shall be labelled with a RFC 4122, UUID optionally proceeded by an identifier of the claim generator and a `:`. An example label for the fictitious ACME claim generator might look like `acme:urn:uuid:F9168C5E-CEB2-4FAA-B6BF-329BF39FA1E4`.

The C2PA Assertion Store is a superbox that shall have a label of `c2pa.assertions` and a UUID of `0x63326173-0011-0010-8000-00AA00389B71` (`c2as`). It shall contain one or more JUMBF Content Type boxes (ISO 19566-5, Annex B) containing the assertion data. These Content Type boxes may be of either CBOR Content Type (`cbor`), JSON Content Type (`json`), Embedded File Content Type (`bfdb`) or UUID Content Type (`uuid`).

The C2PA Claim box shall have a label of `c2pa.claim`, a UUID of `0x6332636C-0011-0010-8000-00AA00389B71` (`c2cl`) and shall consist of a single CBOR Content Type box (`cbor`).

The C2PA Claim Signature box shall have a label of `c2pa.signature`, a UUID of `0x63326373-0011-0010-8000-00AA00389B71` (`c2cs`) and shall consist of a single CBOR Content Type box (`cbor`).

10.1.1.1. Credential Storage

A C2PA Credential Store (VCStore) is a JUMBF superbox that shall contain one or more JSON Content Type boxes (ISO 19566-5, Annex B.4). It shall have a label of `c2pa.credentials` and a UUID of `0x63327663-0011-0010-8000-00AA00389B71` (`c2vc`).

When storing W3C Verifiable Credentials in a VCStore, each one shall be labelled with the value of the `id` field of the `credentialSubject` of the VC itself. Since the `id` is guaranteed to be unique, this ensures that the URI to that credential will be unique.

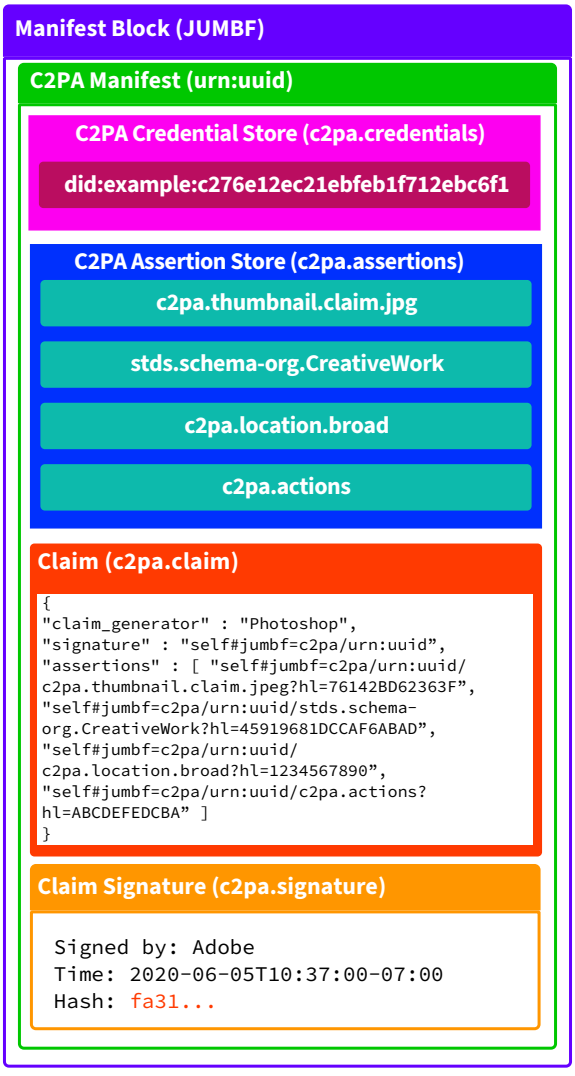


Figure 9. C2PA Manifest Block with Credentials

10.1.1.2. Ingredient Storage

When a C2PA Manifest includes ingredient assertions, and an ingredient contains a C2PA manifest, that manifest must be brought into this asset to ensure that the provenance data is kept intact. Such ingredient manifests are added to

the C2PA Manifest Store as a child of the C2PA Manifest for the asset itself.

Figure 10. C2PA Manifest Block With an Ingredient

10.2. Embedding manifests into assets

10.2.1. Embedding manifests into non-BMFF-based assets

When embedding the C2PA Manifest Store into an asset, the location will vary based on the type of the asset. Here are some well-known types and the location to use:

- JPEG: **APP11** Marker as defined in JPEG XT, ISO/IEC 18477-3. A single marker segment in JPEG-1 cannot be larger than 64K, so it is likely that multiple **APP11** segments will be required, and they will be constructed as per the

JPEG-1 standard and ISO 19566-5, D.2. Writers of multiple segments should write them in sequential order, though they need not be contiguous.

- PNG: An ancillary, private, not safe to copy, chunk type of `'caBX'` (as per PNG, 4.7.2).
- PDF: An embedded file specification (ISO 32000, 7.11.3) which is referenced from the `EmbeddedFiles` NameTree (`/Catalog/Names/EmbeddedFiles`) as well as the value of the **AF** key in the document catalog dictionary. The file specification dictionary shall have an **AFRelationship** key whose value is *C2PA_Manifest*.
- BMFF-based formats: A `'c2pa'` box. Refer to Section 10.2.2, "Embedding manifests into BMFF-based assets" for more information.

Additional locations for other file formats will be added in the future.

10.2.2. Embedding manifests into BMFF-based assets

10.2.2.1. FileTypeBox

A BMFF file that includes an embedded manifest shall include 'c2pa' in the compatible_brands list in the FileTypeBox (`'ftyp'`).

10.2.2.2. The `c2pa` Box

All BMFF-based C2PA assets, whether they are timed (e.g., videos with or without audio tracks), untimed (e.g., still photos) or mixed (e.g., live or animated photos) audiovisual media, shall use a `'c2pa'` box that adheres to the following syntax and semantics:

Syntax

```
aligned(8) class ContentProvenanceBox extends FullBox(`'c2pa'`, version = 0, 0) {
    string box_purpose;
    bit(8) data[];
}
```

Semantics

box_purpose	[indicates purpose of box]
data	[depends on box_purpose]

The box_purpose and fields that depend on it are described below for each box purpose.

NOTE

Regarding unique ids:

There are cases, such as fragmented MP4 (fMP4), where the id for a subset of the asset, such as the track_id field of the `'tkhd'` box, is only locally unique to a subset of the overall asset rather than globally unique to the asset.

Because a globally unique id is required to determine what to hash, a unique id is included. This unique id does not equal any value from the original asset; each value is instead defined when the manifest is created. The unique id is then combined with an associated local id to form an id that's globally unique to the entire asset.

10.2.2.3. Box Containing the Manifest

The box containing the manifest shall appear before the first `'mdat'` box in the file.

The fields in the corresponding box described above shall be set as follows.

box_purpose	For a manifest, this value shall be `manifest`.
data	When box_purpose is `manifest`, the first 8 bytes shall be the absolute file offset to the first `'c2pa'` box with box_purpose equal to `merkle`. If this file contains no such boxes, the first 8 bytes shall be zero. Those 8 bytes shall be followed by the raw manifest bytes followed by zero or more unused padding bytes.

For fragmented mp4 files, an identical `c2pa` box of type `manifest` shall be present in each initialization segment; the manifest must be identical.

10.2.2.4. Auxiliary Boxes for Large and Fragmented Files

Some files have one or more very large `'mdat'` boxes (e.g., large video or image files which may be downloaded and rendered progressively) or large numbers of independent 'mdat' boxes (e.g., fMP4 where each fragment can be downloaded independently).

In these cases, it is unreasonable to require a client to completely download all `'mdat'` box(es) before verifying any portion of the asset. Avoiding that necessity is resolved as follows.

The portion of the manifest containing the BMFF Hash shall include the `merkle` field. Refer to [_hashing_a_bmff_formatted_asset_using_an_exclusion_list] for more information.

For large `'mdat'` boxes that can be verified piecemeal, two or more `'c2pa'` boxes with box_purpose set to `'merkle'` as described below shall be included in the single asset file. They shall follow the last `'mdat'` box in the file. The hash used for a given leaf node in the merkle tree shall be computed over an individual chunk of samples as defined by the `'stco'` or `'co64'` box. All such `'c2pa'` boxes shall occur in the same sequence as the chunks they

hash.

NOTE

EDITORS NOTE

For untimed-media, the `'stco'`/`'co64'` box is not present and thus cannot be used to determine hash division points for large `'mdat'` boxes. For HEIF and AVIF, information in the `'iloc'` ("item location") box can be used for this purpose. Are there other scenarios where an `'mdat'` can be large enough to be worth dividing? If so, what box(es) should be used to decide on division points?

For fragmented mp4 files, one `'c2pa'` box with box_purpose set to `'merkle'` as described below shall be included in each fragment file per `'moof'` / `'mdat'` pair in that file. Each one shall be between the `'moof'` and `'mdat'` box it covers. The hash used for a given leaf node in the merkle tree shall be over all data following the last `'mdat'` box preceding that `'c2pa'` box (or start of file) and preceding the first `'moof'` box following that `'c2pa'` box (or end of file) except data excluded by the exclusion list.

Regardless of how the asset is structured, the fields in the corresponding box described above shall be set as follows.

box_purpose	For an auxiliary box, this value shall be `merkle`.
data	When box_purpose is `merkle`, this value shall contain raw CBOR bytes indicating how to verify a portion of the asset as defined as follows. If there are multiple `'c2pa'` boxes with box_purpose `merkle` for a given merkle tree in a single file, each shall be followed by sufficient padding bytes (zero or more) to make all `'c2pa'` boxes for that merkle tree a fixed size.

NOTE

When there are more than one of these boxes in a single file, i.e., the case where there are large `'mdat'`(s) being verified piecemeal, a fixed size is required in order to enable a progressively downloading client to only download the boxes it needs to begin validation rather than the entire merkle tree. Such a client can download enough of the first of these boxes based on the file offset in the `manifest` to determine if its uniqueId and localId match the 'mdat' it is trying to verify. If they do, it can determine the file offset to the box it needs to verify the current chunk verifying by multiplying the chunk number by that size then download just that box. Otherwise, it can determine the file offset to the beginning of the next merkle tree by multiplying that fixed size by the current merkle tree's total number of leaf nodes, and it can repeat this process until it locates the box it needs. The total download size for this subset of boxes is very small relative to the size of a single chunk.

NOTE

A fragmented mp4 asset may also be stored as a single flat mp4 file with a single `moov` for all tracks and then `moof`/`mdat` pairs for each fragment. Such a file shall be handled as if the individual fragments were moved into their own files, manifest and chunk processing was performed, and then the files were appended together.

10.2.2.4.1. Schema and Example

The CDDL Definition for this type is:

```
; The data structure used to store sufficient information to verify a single 'mdat' box or
; a portion of an 'mdat' box when a Merkle tree is used",
bmff-merkle-map = {
  "uniqueId": int, ; A unique integer used to differentiate local ids
  "localId": int, ; A local id indicating which 'mdat' box this entry pertains to.  This may
not be globally unique
  "location": int, ; Zero-based index into the leaf-most merkle tree row corresponding to
this 'mdat' box or portion of this 'mdat' box
  ? "hashes": [1* bstr], ; An ordered array representing the set of additional hashes
required to reach a hash in the merkle tree specified in the manifest from leaf-most (peer
of this node) to root-most (child of node in manifest).  Note that this array may not be
present, e.g. if the manifest itself contains the leaf-most row of the merkle tree.  Null
hashes are not included in this array.  The algorithm used shall be determined using the
`alg` field from the corresponding entry in the `merkle` field array in the BMFF hash
structure.
}
```

An example in CBOR-Diag is shown below:

```
{
  "uniqueId": 1339,
  "localId": 4402,
  "location": 2203,
  "hashes": [
    b64'TWVub3JhaA=='
  ]
}
```

For files that use a `'tkhd'` box to indicate individual tracks, the localId in the preceding CBOR shall be set to the track_id field of the `'tkhd'` box pertaining to the `'mdat'` being hashed.

NOTE

EDITORS NOTE

For untimed-media (e.g., images) and mixed media (e.g., a single BMFF asset containing both audio/video and an image), a `tkhd` box alone cannot be used to reference the complete set of different `mdat` boxes when there are more than one; it may not be present at all. For scenarios such as these, what box value(s) should be used for localId instead?

10.2.3. Dynamic stream generation

Many adaptive bitrate streaming (ABR) implementations store a single version of an asset, e.g., as a flat MP4 or in another intermediate format, and generate individual asset streams using various codecs, bitrates, etc. at consumption time. As a result, such a server must either hash said streams and create a manifest each time the content is consumed or, if generation is deterministic, create and cache the hashes and manifests once and then embed them at consumption time.

This also means that such a server must have a signing identity of its own that will be trusted by validators or be able

to sign the generated files on behalf of the content creator in a secure and trustworthy manner.

10.2.4. Exclusion List Requirements

For all BMFF files where a `'c2pa'` box includes C2PA information:

```
xpath = "/c2pa"
version = 0
flags = "AAAA"
```

All boxes present that specify absolute or relative file offsets shall be excluded in whole or in part.

The following are examples of entire boxes that shall be excluded (when present).

```
xpath = "/moov/trak/mdia/minf/stbl/stco"
version = 0
flags = "AAAA"
```

```
xpath = "/moov/trak/mdia/minf/stbl/co64"
version = 0
flags = "AAAA"
```

The following is an example of excluding a partial box. (When a `'tfhd'` box is present and its flags has bit `0x000001` set, only the base_data_offset field from the box shall be excluded.)

```
xpath = "/moof/traf/tfhd"
version = 0
flags = "AAAJ"
data = [ { 12, "AQAAAA==" } ]
subset = [ { 16, 8 } ]
```

The flags field is the unpadded base64 encoding of the bytes `0x00 0x00 0x09` indicating that flags `0x000001` and `0x000008` are set. For `tfhd`, this effectively means that the box (including size and type) is 28 bytes long.

- 4 byte size
- 4 byte box type `tfhd`
- 1 byte version
- 3 byte flags
- 4 byte track_id
- 8 byte base_data_offset
- 4 byte default_sample_duration

The data field is set to the unpadded base64 encoding of the bytes `0x01 0x00 0x00 0x00` at offset 12.

This exclusion refers only to a 'tfhd' box with its track_id set to 1 (big-endian). It is excluding only the base_data_offset field which is at offset 16 bytes from the start of the `'tfhd'` box and is 8 bytes long. This field must be excluded because it is a file offset and including it would invalidate the hash once the manifest was embedded into the file.

All other portions of the box are included in the hash, i.e., bytes 0-15 and 25-28. Thus, the 4 byte default_sample_duration field at offset 25 **is** included in the hash. Failing to include that field in the hash would enable the audio/video speed to be increased or decreased without invalidating the hash, thus defeating a stated goal of C2PA.

10.2.5. Timed-media streams that are neither audio nor video

Timed-media streams that are neither audio nor video, such as text streams for captions, that the claim generator wishes to make tamper evident shall be handled the same way as audio and video streams.

10.2.6. External references

Externally referenced content declared inside BMFF boxes, such as in a `'dref'`, `'url'`, or `'urn'` box, that the claim generator wishes to make tamper evident shall **not** exclude the referencing box AND shall include a separate cloud data assertion for each external reference to be hashed.

Chapter 11. Entity Diagram

The following diagram provides a look at how all of the pieces of the C2PA system integrate and relate to each other.

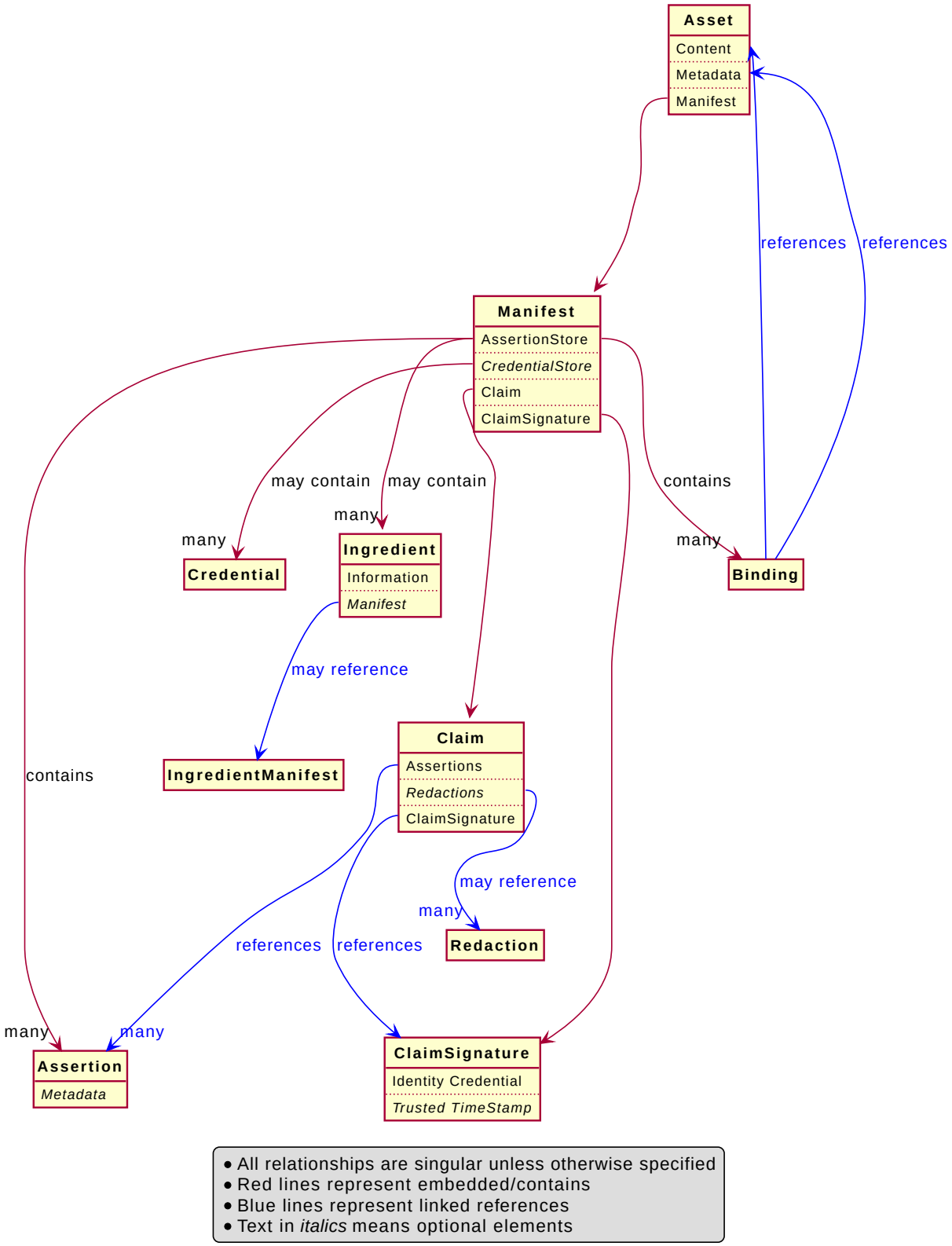


Figure 11. C2PA Entity Diagram

Chapter 12. Cryptography

12.1. Hashing

All cryptographic hashes that are stored in C2PA data structures (e.g., claim or assertions) shall be generated using one of the hash algorithms listed as supported by the C2PA data hash algorithm registry (TBD). This registry will define:

- A list of hash algorithms that are allowed for generating hashes of new content as well as required for validating hashes of existing content (the "allowed" list), and
- A list of hash algorithms that are required to be supported for validating hashes of existing content but are not allowed for generating hashes of new content (the "deprecated" list).

NOTE There is a separate registry for soft binding algorithms.

An algorithm must appear in no more than one list. An algorithm that is instantiated over multiple output lengths will be considered different algorithms for this purpose, and each instantiation must be listed. If an algorithm does not appear in either list, it is forbidden and must not be used. Algorithms can be removed from the lists in order to implement forbidding an algorithm.

This list will establish the allowed algorithms for creating hashes and a string algorithm identifier to be used as the algorithm identifier in the corresponding field. The outputs of hash functions shall be stored as their binary values encoded into CBOR as byte strings (major type 2) with a declared length. Wherever a field contains the output of a hash function, an algorithm identifier string field shall be present within the same structure, or within an enclosing structure, to declare which algorithm was used. A hash algorithm identifier field should be present in exactly one of these places, but if more than one is present within the structure and its enclosing structures, the nearest identifier must be used. Nearest is defined first as an identifier that is a sibling field of the hash value, and then the immediately enclosing structure, up to the root structure.

If the C2PA's data hash algorithm registry has not yet been established, then the registry (the "bootstrap registry") shall contain an empty deprecated list, and the allowed list shall be:

- SHA2-256 ("sha256")
- SHA2-384 ("sha384")
- SHA2-512 ("sha512")

It is recommended that the string identifiers for hash algorithms conform to how they are referred to in common practice although there is no set standard. In particular, in keeping with their description in RFC 6234, the string identifiers for variants of SHA-2 are recommended to be "sha256," "sha384," and "sha512."

When the C2PA data hash algorithm registry is established, the bootstrap registry shall have no effect as if it did not exist. Algorithms not present in the established data hash algorithm registry are not permitted regardless of their

inclusion in the bootstrap registry.

12.2. Digital Signatures

All digital signatures that are stored in a C2PA data structure (e.g., claim or assertion) shall be generated using one of the digital signature algorithms and key types listed as supported by the C2PA digital signature algorithm and key type registry (TBD). COSE groups digital signatures and key types together, and the registry will follow this pattern. This registry will define:

- A list of digital signature algorithms and key types that are allowed for generating signatures of new content as well as required for validating signatures of existing content (the "allowed" list), and
- A list of digital signature algorithms and key types that are required to be supported for validating signatures of existing content but are not allowed for generating signatures of new context (the "deprecated" list).

This list will establish the allowed algorithms and key types by referencing an algorithm identifier from the relevant standards that define algorithms for COSE and their mappings to CBOR identifiers, including but not limited to RFC 8152 and RFC 8230. These standards also specify the hash algorithm used in the signature scheme. Nothing in Section 12.1, “Hashing” shall apply to this use of hash algorithms; if a digital signature algorithm is present in the digital signature algorithm and key type registry, the use of its specified hash algorithm in the signature scheme shall be allowed and followed.

If the digital signature algorithm and key type registry has not yet been established, then the registry (the "bootstrap registry") of digital signature algorithm and key type identifiers shall be an empty deprecated list, and the allowed list shall be as follows. The parenthetical notes are explainers provided only as an aid to the reader.

- ES256 (ECDSA using P-256 and SHA-256)
- ES384 (ECDSA using P-384 and SHA-384)
- ES512 (ECDSA using P-521 and SHA-512)
- PS256 (RSASSA-PSS using SHA-256 and MGF1 with SHA-256)
- PS384 (RSASSA-PSS using SHA-384 and MGF1 with SHA-384)
- PS512 (RSASSA-PSS using SHA-512 and MGF1 with SHA-512)

When the C2PA digital signature algorithm and key type registry is established, the bootstrap registry shall have no effect as if it did not exist. Algorithms not present in the established digital signature algorithm and key type registry are not permitted regardless of their inclusion in the bootstrap registry.

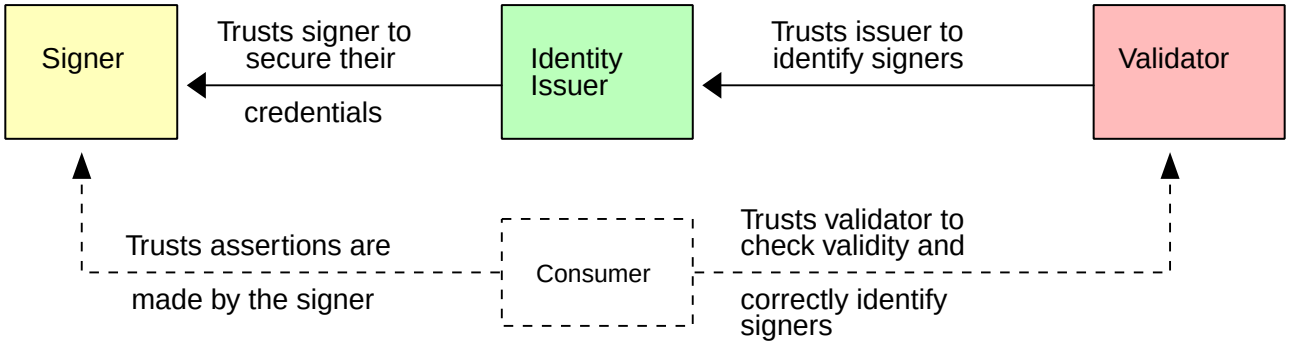
NOTE This list intentionally excludes RS256, RS384, and RS512 which are the RSA variants that use PKCS#1v1.5 instead of PSS as the signature scheme. Though nothing precludes these algorithms from being added to the C2PA supported list when it is established, with no requirement to support legacy code, the problems with PKCS#1v1.5 signature due to Bleichenbacher’s attack are best avoided.

The signature for the CBOR-encoded claim is produced by CBOR Object Signing and Encryption (COSE) as described in RFC 8152 sections 4.2 and 4.4. The signature shall be either a `COSE_Sign1` or `COSE_Sign1_Tagged` structure as stated in the specification of the structure being signed. In the output, it is recommended that the structure being signed is enclosed in the `COSE_Sign1` structure as the payload parameter to avoid the need to define an additional structure to contain the signed version of said structure, and that `COSE_Sign1` structure is used as the envelope. It is allowed to elect a "detached mode" where a custom envelope is defined to carry the signature in which case the contents of the signature field shall be the serialized `COSE_Sign1` (or `COSE_Sign1_Tagged`) structure with the payload field set to nil.

The signature is cryptographically verified as described in RFC 8152 section 4.4.

Chapter 13. Trust Model

13.1. Overview



The above model shows, in yellow, green and red, the three entities specified in the trust model, which is concerned with trust in a signer's identity. In dashed lines, below, is the consumer (who is not specified in the trust model), who uses the identity of the signer, along with other trust signals, to decide whether the assertions made about an asset are true.

13.2. Identity of Signers

Identity in the trust model is the means by which a cryptographic signing key is associated with an actor for the basis of making trust decisions based on any structure (including, but not limited to, claims and manifests) signed with that key. The identity of a signatory is not necessarily a human actor, and the identity presented may be a pseudonym, completely anonymous, or pertain to a service or trusted hardware device with its own identity, including an application running inside such a service or trusted hardware.

All signed structures in C2PA use `COSE_Sign1` structures. The credential should be listed in the unprotected headers. The credential may appear in the protected headers, though as all credential types are themselves signed objects and so carry their own integrity protection, this is not necessary. Regardless, exactly one instance of an identity credential must appear in the union of the protected and unprotected headers. `COSE_Sign1` structures with no credentials, or two or more credentials, must be rejected. Repeating the same credential more than once, including separately in the protected and unprotected headers, is also an instance of two or more credentials and must be rejected.

Each credential type will define the following data to be provided to the validator:

- How the credential is stored in the header value,
- How a trust chain is computed from the signatory to an entry in the validator's trust anchor list,
- The public key in the credential used to validate the signature, and

- The time validity period of the credential.

The name of the header to indicate credential type, how the credential is stored in the header value, and how trust chains are constructed are specified for each credential type below.

13.2.1. X.509 Certificates

X.509 Certificates are stored in a header named `x5chain` draft-ietf-cose-x509. The value is a CBOR array of byte strings, each of which contains the certificate encoded as ASN.1 distinguished encoding rules (DER). This array must contain at least one element. The first element of the array must be the certificate of the signer, and the `subjectPublicKeyInfo` element of the certificate will be the public key used to validate the signature. The `Validity` member of the `TBSCertificate` sequence provides the time validity period of the certificate.

A signer may supply additional certificates, such as intermediate certificate authorities (CAs), to aid a validator in trust chain building. These certificates are included as additional elements of the array, and each is also a byte string containing the DER encoding of the certificate. Each subsequent certificate must be the one used to issue and certify the previous certificate.

13.2.1.1. Anonymous and Pseudonymous Certificates

Anonymous or pseudonymous certificates can be achieved by generating and using a self-signed certificate with no personally identifying information present in the certificate. Applications that generate self-signed certificates for this purpose should endeavor to adhere to the Certificate Profile below, although some requirements do not apply to self-signed certificates. Applications must convey to signers requesting an anonymous certificate that all assets signed with the same certificate are unavoidably linked to one another, and any later exposure of the signer's identity can be imputed to all assets signed with that credential. Self-signed certificates must be explicitly trusted by validators. If compromised, the self-signed nature of such certificates precludes their revocation through in-band means.

Anonymous or pseudonymous certificates can also be achieved by using a trusted third party acting as a certificate authority, which does not publish in certificates personally identifying information. The standards by which these trusted third parties can appear in trust lists are beyond the scope of this trust model, but if adopted, would follow all the same requirements for typical certificates that do identify the signatory. Such trusted third parties should make similar warnings about the ability to link assets signed with a credential in addition to other appropriate education in identity protection.

13.2.1.2. Certificate Profile

All certificates must be signed with an algorithm supported by the C2PA certificate signature algorithm list (TBD). These identifiers are taken from the relevant standards including, but not limited to, RFC 3279, RFC 5758, and RFC 8017 which define their mappings to object identifiers (OIDs).

If the C2PA's certificate signature algorithm list has not yet been established, then the list (the "bootstrap list") shall be as follows:

- ecdsa-with-SHA256

- ecdsa-with-SHA384
- ecdsa-with-SHA512
- sha256WithRSAEncryption
- sha384WithRSAEncryption
- sha512WithRSAEncryption
- id-RSASSA-PSS

When the C2PA list is established, the bootstrap list shall have no effect as if it did not exist. Algorithms not present in the established list are not permitted regardless of their inclusion in the bootstrap list.

All certificates issued by a CA must conform to the following profile and shall be rejected if they do not. Self-signed certificates should conform to the profile below but shall not be rejected for violations if present in an "address book" trust list described below.

- Version must be v3. RFC 5280 section 4.1.2.1
- The `issuerUniqueID` and `subjectUniqueID` optional fields of the `TBSCertificate` sequence must not be present. RFC 5280 section 4.1.2.8
- The Basic Constraints extension must follow RFC 5280 section 4.2.1.9. In particular, it must be present with the `cA` boolean asserted if the certificate issues certificates.
- The Authority Key Identifier extension must be present in any certificate that is not self-signed. RFC 5280 section 4.2.1.1
- The Subject Key Identifier extension must be present in any certificate that is not self-signed and acts as a CA. RFC 5280 section 4.2.1.2
- The Key Usage extension must be present and should be marked as critical. Certificates used to sign C2PA structures must assert the `digitalSignature` bit. The `keyCertSign` bit must only be asserted if the `cA` boolean is asserted in the Basic Constraints extension. RFC 5280 section 4.2.1.3
- The Extended Key Usage extension must be present and non-empty in any certificate where the Basic Constraints extension is absent or the `cA` boolean is not asserted. These are commonly called "end entity" or "leaf" certificates. RFC 5280 section 4.2.1.12
  - The `anyExtendedKeyUsage` EKU (2.5.29.37.0) must not be present.
  - A certificate that signs C2PA structures must be valid for the `id-kp-emailProtection` (1.3.6.1.5.5.7.3.4) purpose. [Editor's note: This is a placeholder. We may want to define our own EKU for this.]
  - A certificate that signs timestamping countersignatures must be valid for the `id-kp-timeStamping` (1.3.6.1.5.5.7.3.8) purpose.
  - A certificate that signs OCSP responses for certificates must be valid for the `id-kp-OCSPSigning` (1.3.6.1.5.5.7.3.9) purpose.
  - A certificate must not be valid for more than one of the three purposes listed above.

- A certificate should not be valid for any other purposes outside of the purposes listed above, but the presence of any other EKUs shall not cause the certificate to be rejected.

13.2.1.2.1. Certificate Trust Chain

The trust chain must be built and validated according to the procedure in RFC 5280 section 6 for the particular purpose required (signing, timestamping, or OCSP signing) and for the appropriate trust list for that purpose. A validator must consider any supplemental certificates provided in building trust chains and may choose to attempt to fetch intermediate certificates on its own. Any failure of that validation algorithm shall mean the chain must be rejected.

Only end entity certificates shall be used to sign C2PA structures or timestamps. A CA certificate must not be used for these purposes. Any chain that terminates in a CA certificate being used to validate a signature, timestamp, or OCSP response must be rejected.

13.2.1.3. Certificate Revocation

X.509 certificates support revocation status queries. C2PA uses the Online Certificate Status Protocol (OCSP) and OCSP stapling to implement revocation. C2PA does not use Certificate Revocation Lists (CRLs).

NOTE

Using CRLs requires downloading the entire list of revoked certificates for each Certificate Authority encountered, which can be time-consuming. Although a CRL could be included in the same way an OCSP response is stapled, the potential size of a CRL relative to an OCSP response also makes this undesirable.

A conforming CA must not use Certificate Revocation Lists (CRLs) nor include a `cRLDistributionPoints` extension (section 4.2.1.13) in the certificate. A conforming CA should include an AuthorityInfoAccess (AIA) extension (section 4.2.2.1) to provide access information for an Online Certificate Status Protocol (OCSP) service operated by the CA.

If the certificate has an AIA header, revocation information shall be stored in an unprotected header of the `COSE_Sign1` structure named `rVals` and shall follow JAdES section 5.3.5.2, with the following modifications:

- The `ocspVals` property shall be present. As C2PA uses CBOR, each item of `ocspVals` shall contain the DER-encoded `OCSPResponse` defined in RFC 6960as a byte string, instead of a Base64-encoded version of the same.
- The `crlVals` property shall not be present. If present, a validator shall ignore its contents, but shall not reject the certificate due to this header being present.
- The `otherVals` property shall not be present. If present, a validator shall ignore its contents, but shall not reject the certificate due to this header being present.

Before signing a claim, if a signer's certificate has the AIA extension, a signer should query the OCSP service indicated therein, capture the response, and store it in an element of the `ocspVals` array of the `rVals` header.

A validator should reject a manifest where the signer's certificate contains the AuthorityInfoAccess extension but the

manifest does not contain an `rVals` header with an `ocspVals` entry that corresponds to the signer's certificate.

NOTE

This is currently a "should" rather than a "must" given that revocation is difficult to implement. Unlike the TLS case, we should not be at the mercy of temporary outages of the OCSP responder; indeed, a temporary failure of the OCSP responder would result in a delay at signing time, as it would delay when the creator could get the OCSP response to include but would not affect validation time. In some cases, a signer may choose to publish without revocation information if the OCSP responder is offline, and there is a time pressure for the signer to publish.

The `rVals` header establishes that the signer's certificate is not revoked if:

- The manifest is also timestamped, and the attested time falls within the `(thisUpdate,nextUpdate)` interval of the OCSP response for the signer's certificate,
- The `certStatus` field of the OCSP response for the signer's certificate is `good`, and
- The signer of the OCSP response is an "authorized responder" as defined by section 4.2.2.2 of RFC 6960.

NOTE

At this time, we are not addressing checking that an authorized responder itself may have been revoked, as described in section 4.2.2.2.1 of RFC 6960. We are also not currently considering potential revocation of intermediate CAs on the path from the signer's certificate to the trust anchor.

13.3. Signer Credential Trust

A validator shall maintain the following trust lists:

- For each credential type:
  - Trust anchors for C2PA signers,
  - Trust anchors for Time Stamping Authorities.

Issuers can appear on more than one list, in which case they are trusted to issue credentials for both purposes. Time stamping authorities must adhere to the requirements in section 2.1 of RFC 3161.

In this section, "user" refers to human actors that are using C2PA-compliant validators in consumption and authoring scenarios.

A validator should allow these trust lists to be configured by the user but should provide default options or offer lists maintained by external parties that the user may opt into.

A validator should provide a list of time stamping authority trust anchors. A validator should allow users to customize this list, but should caution users against doing so, given the heightened security standards trusted timestamping authorities are held to.

A validator may also allow the user to maintain an "address book" of credentials they have chosen to trust based on an out-of-band relationship. For example, a journalist may choose to add sources to their personal address book to

facilitate accepting and validating media with C2PA provenance data attached, even though the sources themselves would have no reason to be on an externally-maintained trust list used broadly by the general public. To facilitate accepting content from such contacts, no violations of the Certificate Profile above will cause claims signed by entries in the address book to be rejected.

13.4. Identity In Assertions

Some assertions (such as `stds.schema-org.CreativeWork`) allow a person's identity to be associated in a defined way with the asset. This identity is purely scoped via the definition in each assertion and does not imply any larger involvement or responsibility for any assertion made in the claim, or the asset itself. All assertions, as stated below, are made by a signer.

13.5. Statements

A validator is a manifest consumer that will produce some validation statements about that asset. The actor consuming the asset, usually through their user agent and its user interface, then has to interpret those statements to arrive at a set of conclusions of their own about the provenance of the asset they are consuming. These conclusions will be drawn from these statements, the set of trust relationships that consumer currently has with the actors in the asset, and the contents of the asset itself.

A validator can make the following true or false statements about the asset they are validating, and no more.

NOTE "not independently verified" is used to indicate that an assertion (like a parent ingredient) has been made by the signer of the claim and has not been verified by another party.

1. The active manifest has not been modified since the active manifest was signed
2. The portions of the asset that are covered by content bindings have not been modified since the active manifest was produced
3. The claim was produced by a claim generator (typically software), and signed by an actor identified in the subject field of the signing credential
4. The assertions of the active manifest are statements by the signer and their contents are not independently verified
5. The assertions of the active manifest have not been modified since the active manifest was produced
6. The assets referred to by ingredient assertions are not (necessarily) available at validation of the active manifest, and therefore their hashes cannot be verified
7. The ingredient assertion may contain a ClaimReview, that indicates the active manifest signer's assessment of the validation state of the ingredient asset's hashes at the time of adding the ingredient
8. The content of ingredient assertions, like all other assertions, is not independently verified

Chapter 14. Validation

The active manifest of an asset is valid only if all the steps in this section are successful. This validation must be completed before a validator presents a successful result to a human user or begins to render any content. Validating content as it is rendered to the user is described in Section 14.9, “Validate the Asset’s Content”.

The figure, Figure 12, “Validating a Claim”, is a visual representation of the process of validating a claim (and its assertions).

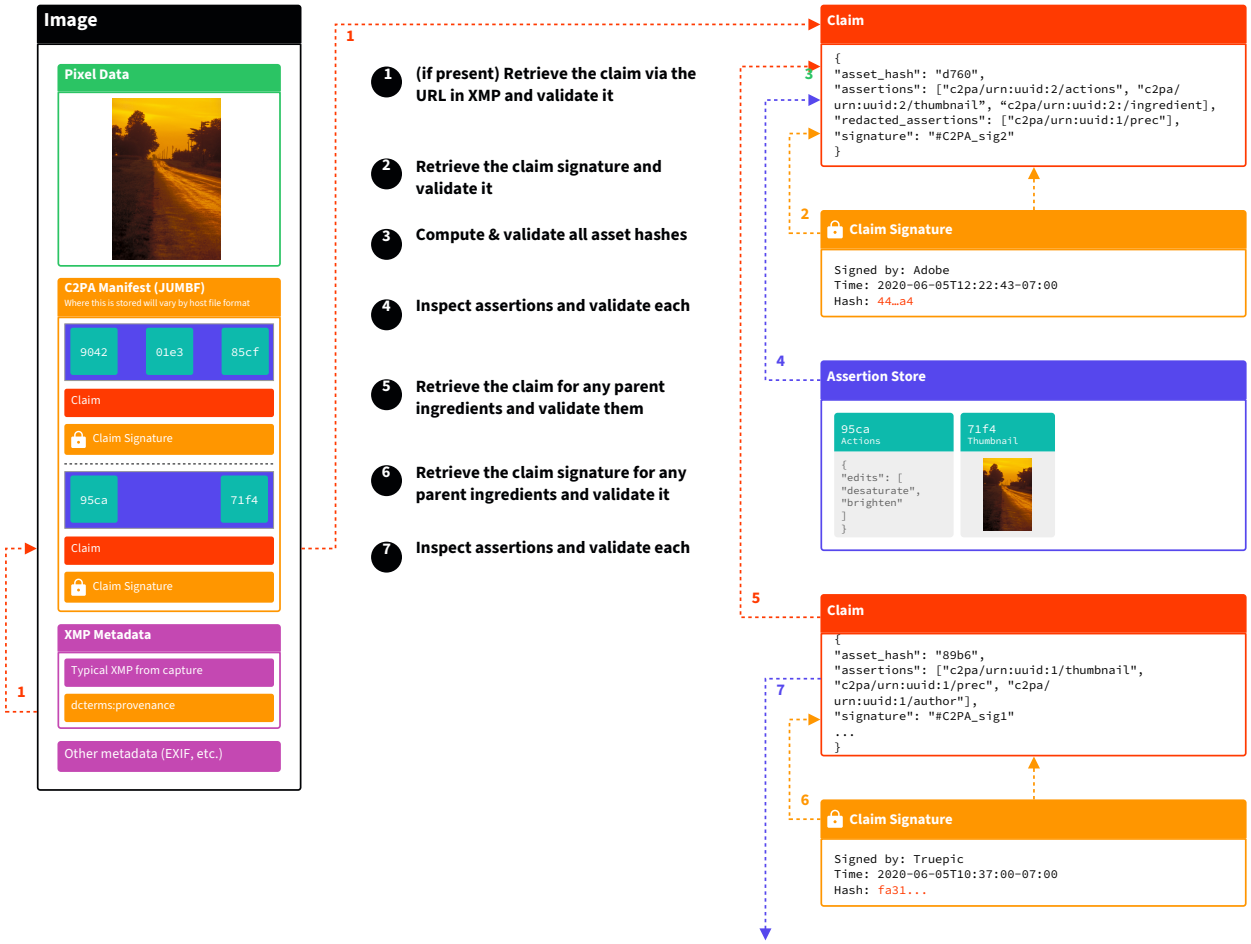


Figure 12. Validating a Claim

14.1. Validate the Claim

If the asset contains XMP, then retrieve the value of the `dcterms:provenance` key from the XMP. If a `dcterms:provenance` key is not present in the XMP, then the asset should be considered as not containing C2PA information (ie. ``no claim''). When present, its value represents the URI reference to resolve to find the active manifest. Resolve the URI reference for the C2PA Manifest, retrieving the data.

For assets that do not contain XMP, the location of the active C2PA Manifest will be stored in some other location as specified in Section 9.3.3, “Referencing the Active Manifest”.

Example 1. Editor's Note

More information about non-XMP-based methods will be specified here as they are developed.

14.2. Validate the Signature

Retrieve the URI reference for the signature from the value of the claim's `signature` field and resolve the URI reference to obtain the COSE signature. The signature must be embedded in the manifest as described in Section 10.1.1, "C2PA Box details". If the signature URI does not refer to a location within the manifest (a `self#jumbf` location), the claim must be rejected. If no such field is present or the URI cannot be resolved, then the claim must be rejected.

Verify that the credential used in the signature is acceptable according to Chapter 13, *Trust Model*. If it is not, then the claim must be rejected. After confirming it is acceptable, validation should proceed according to the specified procedure in Section 12.2, "Digital Signatures". If validation of the signature fails, then the claim must be rejected.

14.3. Validate the Time Stamp (if present)

If the `sigTst` header is not present, the claim is valid if the signer's credential is valid at the current time.

If the `sigTst` header is present, the claim is valid if the `tstTokens` array contains at least one `tstToken` whose `val` property is an RFC3161-compliant `TimeStampToken` which satisfies the following requirements:

- The timestamp contains a message imprint as described in Section 9.3.2.5, "TimeStamps" that matches the claim being validated,
- The time attested by the Time Stamping Authority (TSA) falls within the validity period of the signing credential,
- The attested time falls within the validity period of the TSA's signing certificate,
- A trust chain can be built to an entry in the time stamping authority trust list as described in Section 13.3, "Signer Credential Trust". Locating the TSA's certificate in the timeStampToken is described in RFC 3161 section 2.4.1.

NOTE At this time, the revocation status of a Time Stamping Authority's certificate is neither captured at signing time nor validated at validation time.

14.4. Validate the Credential Revocation Information (if present)

If the signer's credential type does not support revocation status, or the credential's issuer did not provide a method to query its revocation status, the validator presumes the credential is not revoked.

If the signer's credential type supports revocation, and the credential's issuer provided a method to query its revocation status:

- If the `rVals` header is present, its contents should be validated as described in the signer credential type's description in Chapter 13, *Trust Model*.
- If the `rVals` header is not present, and the signer's credential is valid at validation time, a validator may choose to query the credential status method to determine if the credential is currently revoked, which may differ from its status at the time of signing. Querying the credential status method can reveal to an observer the identity of the asset being validated, and so this query is optional. This query will also only reveal if the signer's credential is valid at the current time, which may differ from its status at the time of signing. If the validator chooses to make this query, the response to the query shall determine the credential's status. If the validator does not make this query, the credential shall be presumed not revoked.

NOTE

When a signer's credential is revoked, this does not invalidate manifests that were signed before the time of revocation. The inclusion of the `rVals` header combined with a timestamp provides proof that the signer's credential was valid at the time of signing. Absent this information, because it is not possible to query the historical revocation status of a credential, only its status at validation time, a validator querying in this situation might reject a manifest if the signer's credential was revoked after signing time. Signers are encouraged to include revocation information and time stamps to avoid this possibility.

In all cases, if the credential is deemed revoked, the claim must be rejected.

14.5. Validate the Assertions in the Asset's Claim

Each assertion in the `assertions` field of the claim is a `hashed_uri` structure. If the active manifest's claim is being validated, the `active_redacted_assertions` list is empty. If an ingredient claim is being validated, the `active_redacted_assertions` list is an input to this recursive call. For each assertion, the validator must:

1. If the URI reference in the `url` field is present in the `active_redacted_assertions` list:
    a. If the assertion's label is `c2pa.action`, the claim must be rejected.

        NOTE `c2pa.action` assertions may not be redacted.

    b. Otherwise, the redacted assertion is considered valid, and validation continues to the next assertion.
2. Otherwise, since the assertion is not redacted:
    a. Resolve the URI reference in the `url` field to obtain its data. If the URI cannot be resolved and the data retrieved, the claim must be rejected.
        i. If the URI refers to an external location and there is a temporary failure preventing its retrieval, the claim must not be accepted until all external references can be retrieved. The validator may indicate the temporary failure status and offer the option to render the asset with an unknown provenance state, as it would for an asset without a manifest.
        ii. Compute a hash of the data using the hash algorithm, determined by following the procedure described in Section 12.1, "Hashing":

A. If an `alg` field is present in the `hashed_uri` structure, that determines the hash algorithm.

B. If an `alg` field is not present in the `hashed_uri` structure, an `alg` field must be present in an enclosing structure, and the nearest instance present determines the hash algorithm.

C. If no `alg` field is found in any of these locations, the claim must be rejected.

iii. Compare the computed hash value with the value in the `hash` field. If they do not match, the claim must be rejected.

iv. Otherwise, the assertion is valid and validation continues to the next assertion.

Then, for each element of the claim's own `redacted_assertions` array, if any element of the claim's `assertions` array has a `url` field equal to that value, the claim must be rejected. A claim cannot redact its own assertions, only those of any ingredients.

NOTE

As described in Chapter 7, *W3C Verifiable Credentials*, any proofs present inside a Verifiable Credential are not validated. Like all contents of an assertion, C2PA only guarantees the contents of the credential are integrity-protected.

14.6. Recursively Verifying Integrity of Ingredients

A validator must perform the above validation steps for the asset being presented and its manifest. If any of the above steps conclude the manifest is invalid, that manifest must be rejected.

An asset's manifest may list one or more ingredients. A validator may optionally recurse through any ingredient manifests and verify the hash in the `hashed-uri` reference to check its integrity. There is no requirement that signers of ingredient manifests are trusted by the validator, and building of trust chains of signers of ingredients shall not be attempted. Instead, as the ingredient is included by the signer of the active manifest, and if the signer of the active manifest is accepted per the rules above, ingredient manifests will share in that trust for the purposes of this recursive validation. Applications should not display data from ingredient manifests with failed integrity checks. If the application chooses to display such data, it must flag the display with a warning about the failed integrity check, and that the data cannot be reliably attributed to the ingredient manifest's signer nor to the asset's manifest's signer.

When ingredients are being added to an asset as part of an authoring workflow, the ingredient undergoes full validation. In this scenario, the ingredient's manifest is considered the active manifest for validation purposes, before the ingredient is added to another asset's ingredients.

For consumption scenarios, it is expected that problems with ingredient manifests would be ignored during normal consumption use but may be surfaced as a warning if a user opts to explore the provenance history. Indeed, a consumption application may opt not to do any recursive validation unless the user requests an exploration of the provenance history.

In authoring scenarios, it may be desirable to more prominently raise warnings so that a creator making use of such an asset with a flawed provenance history can make an informed decision of how to proceed.

If a validator chooses to validate ingredients, for each ingredient, it must recursively:

- If the ingredient does not have a `c2pa_manifest` field, the ingredient is accepted. Otherwise:
  1. Create an `active_redacted_assertions` list which is the concatenation of the claim's `redacted_assertions` array with any pre-existing `active_redacted_assertions` list from previous recursive calls. This list is therefore only the redacted assertions from the active manifest's claim and any ingredients along the path to the current ingredient.
  2. Resolve the URI reference in the `url` field to obtain the ingredient claim's manifest. If the URI reference cannot be resolved, the ingredient claim is rejected. If the URI reference refers to an external location, and a temporary failure prevents retrieval of the claim, the claim cannot be accepted until it is retrieved. The validator may indicate this temporary failure status.
  3. Determine the hash algorithm identifier as determined by following the procedure described in Section 12.1, "Hashing":
     a. If an `alg` field is present in the `hashed_uri` structure, that determines the hash algorithm.
     b. If an `alg` field is not present in the `hashed_uri` strcuture, an `alg` field must be present in an enclosing structure, and the nearest instance present determines the hash algorithm.
     c. If no `alg` field is found in any of these locations, the claim must be rejected.
  4. Compute the hash of the ingredient manifest's data using that algorithm.
  5. Compare the computed hash with the value in the `hash` field. If the hashes are not equal, the claim must be rejected.
  6. If the ingredient contains a `metadata` field, and that field contains a non-empty `reviewRatings` field, the ingredient's claim is admitted. An admitted manifest is treated like an accepted manifest, but with an explicit indication that it may contain validation errors that are known to the signer. If the `reference` field is present, its contents are ignored, and the `reviewRatings` (if present and non-empty) still applies to the ingredient's claim. If admitted in this way, the validator must present the contents of the `reviewRatings` field as part of any exploration of the provenance history. Validators should perform full validation if exploration of the provenance history of the ingredient is requested to indicate where there are validation errors.

     NOTE The presence of a non-empty `reviewRatings` is an explicit statement by the signer that they acknowledge and override validation errors in the ingredient's claim itself.

  7. Otherwise, validate the ingredient claim and assertions as described beginning in Section 14.1, "Validate the Claim", except skip establishing signer credential trust, as this is not applicable to ingredients.
     a. When validating the assertions as described in Section 14.5, "Validate the Assertions in the Asset's Claim", provide `active_redacted_assertions` as an input.
     b. The validator may optionally recursively verify the ingredient's ingredients. If it does and those are accepted, the ingredient's claim is accepted. If any are rejected, the ingredient's claim is rejected. If the validator chooses not to recurse further, the ingredient's claim is accepted.

14.7. Inability to access external URIs

During the validation process, a given URI reference may refer to an external location that needs to be resolved. However, there are various reasons that may prevent a validator from obtaining the necessary data at that URL - (a) validation device is offline, (b) service behind the URL is offline, (c) data is no longer present in the external device (e.g., cloud storage system), etc. Such a claim shall not be accepted at any time before all references can be resolved and the entirety of the claim validated, but validation may be deferred or an appropriate status indicating a temporary failure may be returned as suits the needs of the workflow performing validation.

14.8. Visual look of Validation

Here is a visual representation of the process of validating a claim (and its assertions).

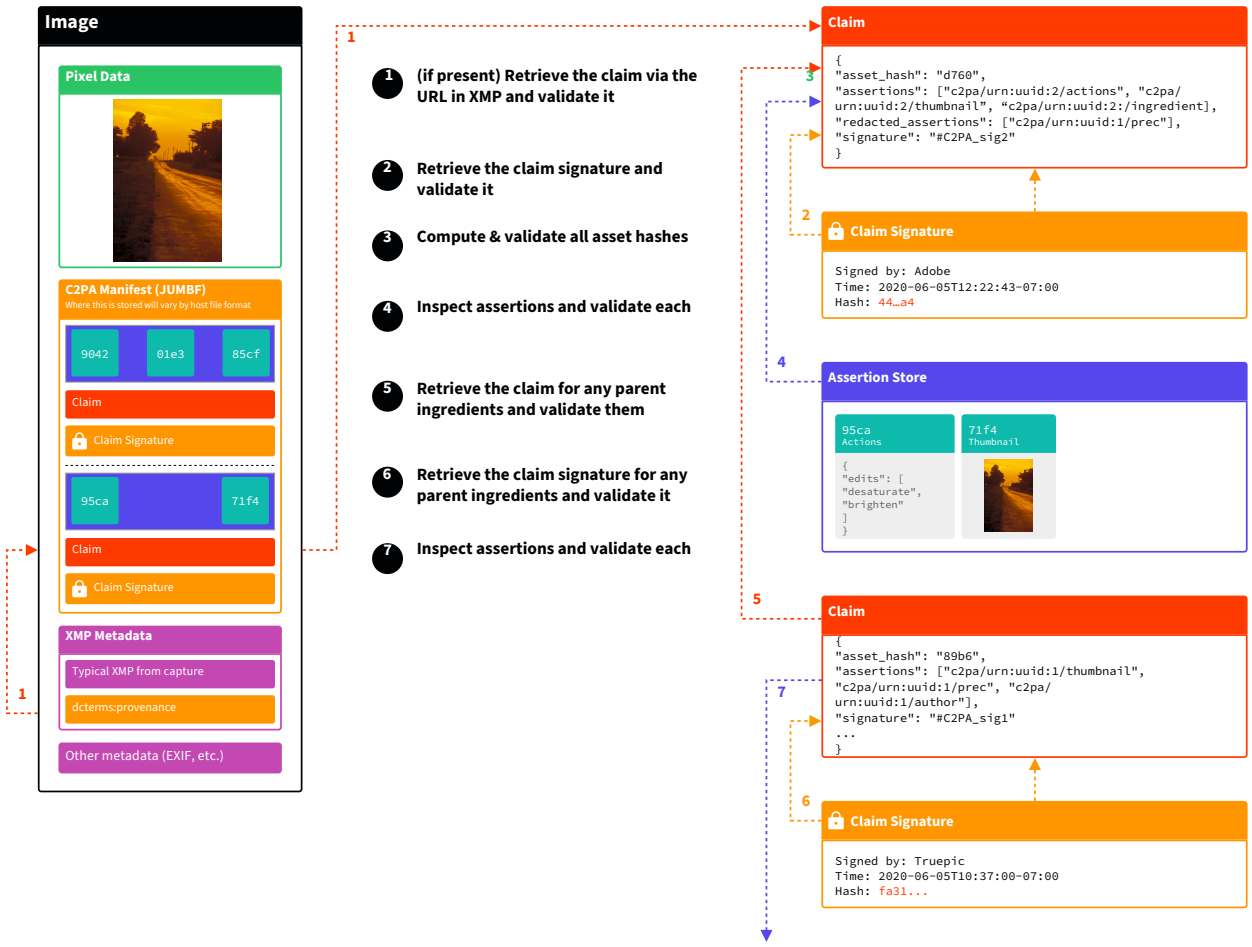


Figure 13. Validating a Claim

14.9. Validate the Asset's Content

For any portions of an asset rendered for presentation to a user, including but not limited to audio, video, or text, the corresponding hard bindings corresponding to the rendered content must be validated in accordance with Section

8.2, “Hard Bindings”. If at any time content fails to be validated, the validator must clearly signal to the user that some of the content does not match the claim, and if possible, should indicate what part of the content did not validate. If any content is absent for which content bindings exist, discovery of this absence is also a validation failure. The validator must continue to report validation has failed, even if later portions of the content validate correctly.

For content that is not wholly available before rendering begins, such as during adaptive bitrate streaming (ABR) and progressive download, absence of not-yet-available portions of content is not considered a validation failure. As the content becomes available, the validator must validate each portion of the content before it is rendered as previously described. In addition, the validator must validate that the sequence of said content is the same as when the manifest was produced. Unless the player has explicitly signalled the validator that a discontinuity is expected (e.g., when the consumer performs a manual seek operation via the UI), the validator must clearly signal to the user that an unexpected discontinuity has occurred whenever the sequence does not match.

For content that is intentionally not being rendered as the claim generator originally intended, such as during fast-forward, rewind, or playback at a different speed, the validator may not be able to validate the content. In this case, the validator must clearly signal to the user that the content cannot be validated during the corresponding operation.

Chapter 15. User Experience

15.1. Principles

C2PA user experience recommendations aim to define best practices for presenting the provenance captured in C2PA systems to consumers. These recommendations are provided as a companion document to this specification (TODO: create link to UX Recommendations).

These recommendations strive to describe standard, readily recognizable experiences that:

- provide asset creators a means to capture information and history about the content they are creating, and
- provide asset consumers information and history about the content they're experiencing, thereby empowering them to understand where it came from and decide how much to trust it.

C2PA consumer user interfaces must be informed by the context of the asset for which they display C2PA data. We have studied 4 primary user groups and a collection of contexts in which C2PA assets are encountered. These user groups are Consumers, Creators, Publishers and Verifiers (or Investigators). To serve the needs of each of these groups across common contexts, exemplary wireframed user interfaces are presented for many common cases. These are recommendations, not mandates, and we expect best practices to evolve.

15.2. Disclosure Levels

Because the complete set of C2PA data for a given asset can be overwhelming to a user, we describe 4 levels of progressive disclosure which guide the designs:

- Level 1: An indication that C2PA data is present and its cryptographic validation status. Should invite the user to learn more.
- Level 2: A summary of C2PA data available for a given asset. Should provide enough information for the particular content, user, and context to allow the consumer to understand to a sufficient degree how the asset came to its current state.
- Level 3: A detailed display of all relevant provenance data. Note that the relevance of certain items over others is contextual and determined by the UX implementor.
- Level 4: For sophisticated, forensic investigatory usage, a standalone tool capable of revealing all the granular detail of signatures and trust signals is recommended.

15.3. Common Applications and Use Cases

TODO: list the use cases that are in scope for the UX Recommendations document.

Chapter 16. Information security

16.1. Threats and Security Considerations

This section provides a summary of information security considerations and processes for technology described in the C2PA core specification. More detailed content will be provided in future releases of C2PA material.

16.1.1. Public review and feedback

Readers of the C2PA specification, including non-members, are encouraged to submit feedback on security considerations documented here via the C2PA feedback process.

16.1.2. Context

Information security is a principal concern of C2PA. C2PA maintains a threat model and security considerations for the C2PA specification. This effort complements other security-related work within C2PA. Associated documentation is currently in development. A draft version of this content is expected to be included in a future public release of C2PA material, and will be referenced here when available.

The C2PA is developing security considerations documentation that includes:

- A summary of relevant security features of C2PA technology
- Security considerations for practical use of C2PA technology
- Threats to C2PA technology and respective treatment of those threats, including countermeasures

16.1.3. Threat modeling process overview

The C2PA builds security into our designs as they are being developed, but also expects that security design and threat modeling will continue as the system, ecosystem, and threat landscape evolve.

To this end, the C2PA uses a focused threat modeling process to support development of a strong security and privacy design. Outcomes of the effort directly support development of explicit threats and security considerations documentation, but also facilitate security thinking throughout the design process.

The threat modeling process combines synchronous (live) threat modeling sessions with focused groups of SMEs with asynchronous development of content. The number of attendees in each synchronous session is kept small to promote efficient discussions, but all members of the C2PA have the opportunity to participate via either modality.

Like other security activities, we expect our threat modeling process to evolve with the C2PA ecosystem. Process documentation is considered a guide rather than a strict directive on how threat modeling works within the C2PA.

16.1.3.1. References

A variety of references and experiences are used to inform threat modeling and related security activities for the C2PA. This section provides a subset of public documents for reference.

- IETF on security considerations
- IETF on privacy considerations (guidelines)
- W3C security and privacy self-review questionnaire
- OAuth2 threat model (example)
- Threat Modeling: Designing for Security
- OWASP Threat Modeling
- Microsoft Threat Modeling

16.1.4. Security considerations

This section summarizes considerations and practical guidance for securing C2PA implementations.

IMPORTANT	This content does not specify C2PA requirements and is not intended to be comprehensive for all potential C2PA deployments.

16.1.4.1. Protecting claim signing keys

In practice, C2PA claim signing keys will be issued to systems that perform claim signing operations. These systems may make these operations available to end users and/or be deployed to user-owned platforms (e.g. mobile phones). Issuance or disclosure of claim signing keys to malicious actors enables attackers to create claim signatures on arbitrary assets using the compromised identity. The resulting manifests are valid in terms of the C2PA specification, but effectively allow for spoofing provenance.

Thus, it is important that systems that manage C2PA claim signing keys adhere to security and key management best practices. This includes leveraging platform-specific features (e.g. hardware security modules, cloud key management services), minimizing key reuse, and revoking keys when compromise is suspected. For more information on key management, see the NIST Key Management Guidelines.

Likewise, it is recommended that key issuance processes be robust against exploitation and misuse. The CAB Forum Baseline Requirements provide a working example of security requirements for signing credential issuers.

16.1.4.2. Securing claim generation and signing operations

Some C2PA claim generation and signing systems may be exposed to untrusted users. Exploitation or misuse of these systems may allow attackers to create claim signatures on arbitrary assets using identities provided by the system. The resulting manifests are valid in terms of the C2PA specification, but effectively allow for spoofing provenance. The impact of such an attack may be amplified if identities are shared between users, and/or if the attack goes undetected

for an extended period of time.

C2PA claim generation and signing systems should consider industry best practices for information security, secure development and operation, and anti-abuse practices, including leveraging available platform-specific features for deployment (e.g. Android SafetyNet, Apple DeviceCheck and AppAttest).

Likewise, it is highly recommended that all signing credentials include revocation information, so that they can readily be revoked in case of suspected compromise.

16.1.4.3. Validation security practices

A manifest consumer that is performing validation (e.g. a web browser) may detect and mitigate attempted compromise of C2PA manifests, or complete removal of C2PA manifests. It is recommended that manifest consumers consider forthcoming C2PA Ux guidance, retrieval of decoupled manifests via soft bindings when appropriate, and other forthcoming C2PA recommendations to mitigate the impact of these types of attacks.

Special care should be taken when implementing validators. Like other software that processes untrusted input, validators may be the target of memory safety attacks, parser attacks, request forgery attacks against adjacent systems (e.g. when retrieving remote content or decoupled manifests), information leaks (e.g. via OCSP queries), denial of service attacks, and so on. Thus, it is important that these validators adhere to secure development and operations practices associated with their respective execution environment.

16.1.4.4. Content binding security

C2PA recommends using embedded, hard-bound manifests whenever possible. Use of hard-bound manifests prevents collision-based attacks associated with soft bindings described in forthcoming threat model documentation. Likewise, use of embedded manifests can help mitigate issues related to separate retrieval of manifests, as well as TOCTOU issues.

However, decoupled manifests and soft bindings can be used to mitigate some threat scenarios. They might also be useful in non-malicious cases, e.g. where platforms incidentally strip C2PA metadata. In these cases, special care must be taken to adhere to the guidance provided in the forthcoming decoupled manifest documentation.

16.2. Harms, Misuse, and Abuse

16.2.1. Considerations

The harms, misuse, and abuse assessment is an ongoing process. The information presented below should not be considered the end result of a comprehensive evaluation, but a basis for broader, ongoing, and more profound discussions, centering on impacted communities, that could lead towards the mitigation of potential abuse and misuse and the protection of human rights.

There are two critical aspects of the approach:

- **Ongoing**: The harms, misuse, and abuse assessment and identification of guidance, potential mitigations or

compliance areas, system-change requirements and no-go areas accompanies the design, development, implementation and use stages of the C2PA standard.

- **Multi-disciplinary and diverse**: The harms, misuse, and abuse assessment is a collaborative effort that includes multi-disciplinary experts and a broad range of stakeholders with lived, practical and technical experience of the issues and from diverse geographical locations, cultural backgrounds and individual identities.

16.2.2. Introduction

The harms, misuse, and abuse assessment aims to identify how the C2PA specifications could be abused and cause harms, threats to human rights, or disproportionate risks to vulnerable groups globally as per the Guiding Principles for C2PA designs and specifications. Further, as per the Guiding Principles, the standard is being reviewed to:

- Anticipate and mitigate potential abuse and misuse;
- Address common privacy concerns of its users; and
- Consider the needs of users and stakeholders throughout the world.

The following subsections describe the framework and the process carried out to date, followed by the methodology, an overview of the assessment and an outline for public review and feedback. Due diligence actions will be developed parallel to the public review and feedback process.

16.2.3. Harms, Misuse, and Abuse Framework

Harms modelling focuses on analyzing how a sociotechnical system might negatively impact users and other stakeholders. Modelling harms systematically requires combining knowledge about a system architecture and its user affordances, with historical and contextual evidence about the impact of similar existing systems on different social groups. This combined information frames the ability to comprehensively anticipate harm.

Harms modelling considers the ramifications of a technological system both from the perspective of the technology developers as well as users and non-user stakeholders. In other words, harms modelling considers what kinds of harms may result from the configuration of a system as well as what kinds of harms may result from both its intended use and unintended use. It is necessary to combine all of these considerations to achieve a comprehensive perspective on potential harms, particularly on those that may be unanticipated by system developers but highly evident to disproportionately impacted social groups. Following principles from justice-oriented technology development and design justice, it is essential to include comprehensive and ongoing consultations with communities likely to be impacted by the specification, and to place emphasis on those who already face similar systemic harms.

In designing our harms modelling approach, we drew inspiration from different approaches to technology impact assessment, including the fields of value-driven design, human rights due diligence, security-focused threat modelling frameworks, and harms modelling methodologies. After a review of potential methodologies, it was determined that adapted versions of Microsoft's Harms Modelling Framework and BSR's Human Rights Due Diligence Assessment would be used to guide the harms modelling process for C2PA. We also collaborated with colleagues conducting parallel exercises in threat modelling exercises as part of the Technical Working Group and engaged with the User

Experience research and Decoupled Manifest Task Forces within C2PA. Below we present some of the modifications we made from existing frameworks for technology impact assessment:

- **Human rights-focused harm taxonomy**: We wanted to ensure more well-established human rights, privacy and security concerns were analysed as elements of broader forms of harm around social inequality and discrimination, and in relation to issues potentially affecting the particular users and stakeholders of the C2PA (such as media entities, citizen journalists, and human rights defenders). For this reason we modified the harm taxonomy particular to the Microsoft Harms Modelling Framework to reflect these issue intersections, stakeholders, and users. The reader will note that in this iteration, some harm taxonomy categories are broader than others. This reflects the fact that there is significant overlap between categories and using both broad and narrow categories helped us to consider a range of potential harms/misuses/abuses.
- **Temporality**: We found it important to analyze harms and impacts not as a static snapshot in time but as an ongoing process with particular considerations for every stage of technological design, development and use (and potentially non-use). This is reflected in the following scenarios of analysis: 1) Initial Adoption; 2) Wide Adoption and 3) Ongoing Maintenance. These scenarios are explained further in the Harms, Misuse, and Abuse Initial Assessment section.
- **Assigning values for severity, scale, likelihood, frequency and impact**: In this initial stage of analysis, we have conducted an internal process to understand severity, scale, likelihood, frequency and impact of potential harms. We have done so in consultation with issue experts within the C2PA and based on C2PA member WITNESS's work on trade-offs and risks within authenticity and provenance infrastructure (see Ticks or It Didn't Happen: Confronting Key Dilemmas in Building Authenticity Infrastructure for Multimedia). However, we see this exercise as an important starting point for feedback rather than a final analysis. Further stages of harm analysis will be conducted in consultation with outside groups, particularly from communities with lived, practice and expert knowledge, and those who will be disproportionately impacted by harms and are often most excluded from design. This analysis should be ongoing, considering that the degree of severity, likelihood, and impact will likely change and become more evident after the technology is deployed.
- **Considering accountability**: Acknowledging that ethical analyses and threat modelling processes are sometimes done behind closed doors, we find it important to emphasize that our harm analysis should be inclusive and it should guide future processes related to governance and compliance.

16.2.4. Methodology

There are three phases to our methodology. These phases do not reflect a chronological order, they frame specific processes that will need to be continuously iterated as more actors join the discussion and analysis.

16.2.4.1. Phase I: Purposes, Use-cases, Users and Stakeholders

Phase I includes defining the purposes of the technology, its use-cases and stakeholders as it pertains to the harms, misuse and abuse assessment. As with other parts of the C2PA standards development, we began with the Purposes/Use-Cases/Users/Stakeholders from two initiatives, the Content Authenticity Initiative and Project Origin and expanded to other potential scenarios. Some of the questions we were attempting to answer were:

- **Purposes**: What problem will be solved? For who? What new capability will be possible? For who?

- **Use-cases**: What will the C2PA standard be used for? What context will the C2PA standard likely be used in?
- **Users/Actors**: Who will directly interact with the C2PA standard?
- **Stakeholders**: Who will be impacted by the use of the C2PA standard including non-users?

16.2.4.2. Phase II: Harm Taxonomy and Assessment

In Phase II, we reviewed and adapted Microsoft's taxonomy of harm to better reflect the context and implications of the C2PA standard and inform it via the other frameworks noted above. This process was intertwined with the actual assessment of the identified harms. The guiding questions in this phase were:

- How could people be harmed by the use of C2PA? What use-cases are most likely to cause harm? To whom?
- What use-cases are most likely to cause harm? To whom?
- How could a misuse or abuse of C2PA lead to harm? Who would be affected?
- What contextual evidence from either an existing technology or societal phenomenon either provides direct evidence of this harm or harm in a related context?
- What is the severity, scale, frequency, likelihood, and disproportionate impact on vulnerable groups of a particular potential harm?

16.2.4.3. Phase III: Due Diligence Actions

Phase III is aimed at mitigating potential abuse and misuse, and offering considerations and guidelines for the protection of human rights and for the optimization of the benefits that prompted the development of the C2PA standard. The questions that will guide this phase are:

- How could the C2PA standard be designed to prevent harmful impacts?
- How could the C2PA standard be built to protect human rights?
- What guidance, compliance requirements or technical steps can address these?

16.2.5. Harms, Misuse, and Abuse Initial Assessment - Overview

The C2PA standard is still in the design stage, and reflects a system specification not a specific product, so the potential harms identified thus far reflect system-level considerations that may not be relevant for all products built using these specifications.

In an effort to establish a common basis for an analysis and to guide our internal and now public discussions, we propose some scenarios based on three temporal stages of the development and adoption cycle of the C2PA standard. The assessment of the identified harms responds to each one of these scenarios.

Scenario 1: Initial Adoption. In this scenario, we assume that the tool will be deployed by a few key actors across multiple industries. These actors will be primarily, though not exclusively, members of the C2PA. Some of these early adopters are actors with significant influence over their respective industries, and we assume that their example and authority could lead to a scenario of wide adoption.

Scenario 2: Wide Adoption. We assume for this scenario that the C2PA standard could be widely used at a global scale, and that it will be a credible reference of the authenticity and provenance of digital assets. In this scenario, it would be more widely used in social media platforms, by a diversity of media producers and be discussed in legislation or regulation. Despite its widespread use, there would continue to be many actors across different industries, vulnerable groups and geographic locations that do not/cannot use the standard.

Scenario 3: Ongoing maintenance. This scenario crosscuts through the previous two, and reflects the issue of continuous improvement and adaptation of the specification as a response to a dynamic context and threat landscape.

The table below lists the identified harms and classifies them under their respective category and type of harm. In the upcoming versions of the specifications, each harm will be analyzed to determine its severity, scale, likelihood and frequency, in addition to the disproportionate risks to vulnerable groups globally.

Table 1. Identified Harms (version 0.7)

Category	Type of Harm	Harm
Risk of injury	Emotional or psychological distress; Physical harm	Misinformation and Malinformation
Denial of consequential services	Opportunity loss	Educational discrimination
		Digital divide/technological discrimination (1)(2)
		Journalistic Freedom and Independence
		Loss of choice/network and filter bubble
	Economic loss	Devaluation of individual expertise
		Differential pricing for goods and services
		Increased abuse of systems of creative ownerships
		Creative ownership impersonation

Category	Type of Harm	Harm
Infringement on human rights	Dignity loss	Public shaming, malinformation and targeted exposure and harassment
	Liberty loss, discrimination and due process	Augmented Policing and Surveillance
		Loss of Effective Remedy
	Privacy loss	Interference with private life
		Reduction in options for anonymity and pseudonymity
		Never forgotten
	Constraints on Freedom of Expression	Inability to freely and fully develop personality and creative practice
		Enforcement of extralegal or restrictive laws on freedom of expression
	Freedom of Association, Assembly and Movement	Forced association (Requiring participation in the use of techno logy or surveillance to take part in society)
		Loss of freedom of movement or assembly to navigate the physical or virtual world with desired anonymity
	Environmental impact	High energy consumption

Category	Type of Harm	Harm
Erosion of Social and Democratic Structures	Manipulation	Misinformation
	Over-reliance on systems	Overconfidence in technical signals
	Amplification of power inequality	Social detriment
		Journalistic plurality and diversity

16.2.6. Public Review and Feedback

Recognizing the limitations and biases of C2PA members and to ensure feedback on harm, misuse and abuse scenarios and responses, we now solicit input from as many voices as possible, particularly from people and groups across the globe that may consider themselves likely to be impacted by the implementation of this standard. This feedback should center on communities with lived, practical and technical experience of the impact of similar technologies, as well as communities most likely to experience potential harms and that are often excluded from technology design and implementation decision-making.

16.2.7. Due Diligence Actions

The aim of due diligence actions is to mitigate potential abuse and misuse, to offer considerations and guidelines for the protection of human rights, to identify compliance or non-compliance standards, and ensure the optimization of the benefits in terms of trust in media, user control and transparency that prompted the development of the C2PA standard.

Due diligence actions will be developed parallel to the public review and feedback process.

Chapter 17. C2PA Standard Assertions

17.1. Introduction

This section of the document lists the standard set of assertions for use by C2PA implementations, describing their syntax, usage, etc. To keep things simple, all example JUMBF URIs have been shortened for illustrative purposes - full URIs are necessary in the actual data.

All C2PA standardized assertions use the JSON JUMBF content type, the CBOR JUMBF content type, or the Embedded File content type from ISO 19566-5:AMD-1. Entity-specific assertions can be any of those, any of the other JUMBF content types from ISO 19566-5, B.1 (such as XML) or may create its own (as per the instructions in ISO 19566-5, Table B.1). The Codestream content type shall not be used for a C2PA assertion.

NOTE CBOR is not currently defined in 19566-5, but an upcoming update will define the type as `cbor`.

Unless otherwise mentioned, all assertions documented in this standard set of assertions shall be serialized as CBOR. For all assertions are of type CBOR, their schemas shall be defined using CDDL. For those defined using JSON, their schemas shall be defined using JSON Schema.

All assertions shall have a label as described in Section 5.2, “Labels” and shall be versioned as described in Section 5.3, “Versioning”.

17.2. Use of CBOR

All CBOR encoded data in a C2PA manifest shall comply with the "Core Deterministic Encoding Requirements" of CBOR.

When converting JSON data, based on one of the JSON Schemas documented here, to CBOR for the purposes of including as a valid C2PA assertion, the following shall be done:

- A JSON `null` shall be serialized as Null (major type 7, additional information 22).
- A JSON `false` shall be serialized as False (major type 7, additional information 20).
- A JSON `true` shall be serialized as True (major type 7, additional information 21).
- A JSON number shall be serialized either as an integer (major type 0 or 1) or a floating-point value (major type 7, additional information 25 through 27).
- A JSON string that does not have an alternate semantic type (e.g., date-time or uri) shall either be serialized as a fixed-length UTF-8 string (major type 3) without any null termination, if human consumable, or a byte string (major type 2) if not. For Base64-encoded strings, unless specified otherwise, they shall be encoded in CBOR as a byte string (major type 2) after decoding from Base64.
- A JSON string that is identified as a `date-time` shall be serialized with tag 0 (major type 6)

- A JSON string that is identified as a `URI` shall be serialized with tag 32 (major type 6)
- A JSON array shall be serialized as an array (major type 4).
- A JSON object shall be serialized as a map (major type 5), with each key being serialized as a UTF-8 string (major type 3).
- If a JSON object contains a key that is defined (by a specific schema) as optional, then either the key with an associated value shall be included in the map or neither its key nor any value shall be included in the map.
- Indefinite-length items (additional information 31) shall not be used. Accordingly, break codes (major type 7 and additional information value 31) shall not be used.
- A JSON `null`, a JSON array of length zero, nor a JSON string of length zero shall be used unless they are (a) the value of a required (by a specific schema) key in a JSON object or (b) there are explicitly defined semantics (by a specific schema) for their usage.

NOTE The above list is based on an inversion of section 6.1 of CBOR with additions from C2PA best practices.

17.3. Metadata About Assertions

In many cases, it is useful or even necessary to provide additional information about an assertion, such as the date and time when it was generated or other data that may help manifest consumers to make informed decisions about the provenance or veracity of the assertion data. Below shows the core schemas used inside other assertions.

In CDDL:

```
;Describes additional information about an assertion, including a hashed-uri reference to
it. We use a socket/plug here to allow hashed-uri-map to be used in individual files without
having the map defined in the same file
$assertion-metadata-map /= {
  ? "reviewRating": [1* rating-map], ; Ratings given to the assertion (may be empty)
  ? "dateTime":  tdate, ; The ISO 8601 date-time string when the assertion was
created/generated
  ? "reference": $hashed-uri-map, ;hashed_uri reference to another assertion that this
review is about
  * $$assertion-metadata-map-extension
}

$source-type /= "signer"
$source-type /= "claimGenerator.REE"
$source-type /= "claimGenerator.TEE"
$source-type /= "localProvider.REE"
$source-type /= "localProvider.TEE"
$source-type /= "remoteProvider.1stParty"
$source-type /= "remoteProvider.3rdParty"
$source-type /= "humanEntry.anonymous"
$source-type /= "humanEntry.credentialed"


source-map = {
  "type": $source-type, ; A value from among the enumerated list indicating whether the
source of the assertion is a claim generator running in a rich execution environment (REE),
a claim generator running in a trusted execution environment (TEE), a local data provider in
REE (e.g. the location API from a mobile operating system), a local data running in a TEE
(e.g. a trusted location trusted app from a chipset vendor), a remote data provider such as
a server (e.g. Google's geolocation API service), entry by a human who wishes to remain
anonymous, or a human who is credentialed with a Verifiable Credential that's included in
the asset.
  ? "details": tstr .size (1..max-tstr-length), ; A human readable string giving details
about the source of the assertion data, e.g. the URL of the remote server that provided the
data
  ? "credentials" : [1* $hashed-uri-map] ; array of hashed_uri references to W3C Verifiable
Credentials
}

int-range = 1..5

$review-code /= "actions.unknownActionsPerformed"
$review-code /= "depthMap.sceneMismatch"
$review-code /= "thumbnail.primaryMismatch"
$review-code /= "stds.iptc.location.inaccurate"
$review-code /= "stds.schema-org.CreativeWork.misattributed"
$review-code /= "stds.schema-org.CreativeWork.missingAttribution"

rating-map = {
  "value":  int-range, ; "A value from 1 (worst) to 5 (best) of the rating of the item"
  ? "code": $review-code, ; A label-formatted string that describes the reason for the
rating
  ? "explanation": tstr .size (1..max-tstr-length), ; A human readable string explaining why
the rating is what it is
}
```

In JSON Schema:

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
```

```
  "$id": "http://ns.c2pa.org/assertion-metadata/v1",
  "type": "object",
  "description": "Assertion that describes additional information about an assertion,
including a hashed-uri reference to it",
  "examples": [
    {
      "dataSource": {
        "type": "localProvider.REE",
        "details": "Dilip's Photo Editor for Windows v5.6"
      },
      "reviewRatings": [
        {
          "value": 2,
          "code" : "actions.unknownActionsPerformed",
          "explanation": "A 3rd party filter was used."
        }
      ],
      "dateTime": "2021-06-28T16:34:11.457Z"
    },
    {
      "reference": {
        "url": "self#jumbf=c2pa/acme:urn:uuid:F9168C5E-CEB2-4faa-B6BF-
329BF39FA1E4/c2pa.assertions/stds.schema-org.CreativeWork",
        "alg": "sha256",
        "hash": "hoOspQQ1lFTy/4Tp8Epx670E5QW5NwkNR+2b30KFXug="
      },
      "reviewRatings": [
        {
          "value": 1,
          "code": "stds.schema-org.CreativeWork.missingAttribution",
          "explanation": "Producer Thomas Chu was not credited."
        },
        {
          "value": 1,
          "code": "stds.schema-org.CreativeWork.misattributed",
          "explanation": "Editor of asset is Aswhini Viswanathan, not Stacey Higgs."
        }
      ]
    }
  ],
  "definitions": {
    "ACTOR": {
      "type": "object",
      "properties": {
        "credentials": {
          "type": "array",
          "description": "An array of hashed uris to W3C Verifiable Credentials",
          "minItems": 1,
          "items": {
              "$ref": "http://ns.c2pa.org/hashed-uri/v1",
            "description": "hashed-uri reference to a W3C Verifiable Credential (VC)
associated with the person or organization who entered the assertion content."
          }
        }
      },
      "required": ["credentials"],
      "additionalProperties": false
    },
    "DATASOURCE": {
      "type": "object",
      "properties": {
        "type": {
          "enum": [
```

```
            "signer",
            "claimGenerator.REE",
            "claimGenerator.TEE",
            "localProvider.REE",
            "localProvider.TEE",
            "remoteProvider.1stParty",
            "remoteProvider.3rdParty",
            "humanEntry.anonymous",
            "humanEntry.credentialed"
          ],
          "description": "A value from among the enumerated list indicating whether the
source of the assertion is a claim generator running in a rich execution environment (REE),
a claim generator running in a trusted execution environment (TEE), a local data provider in
REE (e.g. the location API from a mobile operating system), a local data running in a TEE
(e.g. a trusted location trusted app from a chipset vendor), a remote data provider such as
a server (e.g. Google's geolocation API service), entry by a human who wishes to remain
anonymous, or a human who is credentialed with a Verifiable Credential that's included in
the asset."
        },
        "details": {
          "type": "string",
          "description": "A human readable string giving details about the source of the
assertion data, e.g. the URL of the remote server that provided the data"
        },
        "actors": {
          "type": "array",
          "description": "An array of the actors that undertook this action.",
          "minItems": 1,
          "items": {
            "type": "string",
            "minLength": 1,
            "$ref": "#/definitions/ACTOR",
            "description": "list of actors"
          }
        }
      },
      "anyOf": [
        {
          "not": {
            "properties": {
              "type": {
                "const": "humanEntry.credentialed"
              }
            },
            "required": [
              "type"
            ]
          }
        },
        {
          "required": [
            "actors"
          ]
        }
      ],
      "required": [
        "type"
      ],
      "additionalProperties": false
    },
    "RATING": {
      "type": "object",
      "properties": {
```

```
        "value": {
          "type": "integer",
          "minimum": 1,
          "maximum": 5,
          "description": "A value from 1 (worst) to 5 (best) of the rating of the item"
        },
        "code": {
          "enum": [
            "actions.unknownActionsPerformed",
            "depthMap.sceneMismatch",
            "thumbnail.primaryMismatch",
            "stds.iptc.location.inaccurate",
            "stds.schema-org.CreativeWork.misattributed",
            "stds.schema-org.CreativeWork.missingAttribution"
          ],
          "description": "A label-formatted string that describes the reason for the rating"
        },
        "explanation": {
          "type": "string",
          "minLength": 1,
          "description": "A human readable string explaining why the rating is what it is"
        }
      },
      "required": ["value"]
    }
  },
  "properties": {
    "dataSource": {
      "$ref": "#/definitions/DATASOURCE",
      "description": "A description of the source of the assertion data, selected from a
predefined list"
    },
    "reviewRatings": {
      "type": "array",
      "description": "An array of review ratings",
      "minItems": 1,
      "items": {
        "$ref": "#/definitions/RATING",
        "description": "The rating given to the assertion"
      }
    },
    "dateTime": {
      "type": "string",
      "minLength": 1,
      "format": "date-time",
      "description": "The ISO 8601 date-time string when the assertion was
created/generated"
    },
    "reference": {
      "$ref": "http://ns.c2pa.org/hashed-uri/v1",
      "description": "hashed_uri reference to another assertion that this review is about"
    }
  },
  "required": [],
  "additionalProperties": true
}
```

In most cases, this assertion specific metadata will appear directly inside of other assertions (e.g., ingredients) as the value of their `metadata` field. However, assertions not in JSON or CBOR, such as thumbnails, will need their information stored as a separate, independent, `assertion metadata` assertion. The label for the `assertion`

`metadata` assertion is `c2pa.assertion.metadata`.

17.3.1. Details

17.3.1.1. The `dataSource` object

This `dataSource` field is an optional field that allows the claim signer to inform downstream manifest consumers about the source from which the assertion contents originated. If no `dataSource` is provided for a given assertion, the `dataSource` is considered to be the `Signer`.

NOTE By default, all assertions are sourced to the Signer, as the Trust Model is rooted in trust of the Signer. Where a different source is indicated, it will be a useful Trust Signal to a manifest consumer.

The value of the field is a `dataSource` object that is composed of three fields: `type`, `details`, and if applicable, `credential`.

The dataSource `type` field defines the type of the dataSource. It is assembled with labels in the format described in Section 5.2, "Labels". The value can be one of the following specification-defined values, or entity-specific labels can be used as an extension mechanism.

Value of `type`	Meaning
`signer`	The assertion contents came from the Signer
`claimGenerator.REE`	Assertion contents came from a claim generator running in a rich execution environment (REE), such as a desktop or mobile operating system (e.g. Windows, iOS, or Android)
`claimGenerator.TEE`	Assertion contents came from a claim generator running in a trusted execution environment (TEE), such as a trusted OS (e.g. Google's Trusty TEE or Trustonic TEE)
`localProvider.REE`	Assertion contents came from a data source running in an REE on the same physical computing device as the claim generator (e.g. iOS camera APIs)
`localProvider.TEE`	Assertion contents came from a data source running in a TEE on the same physical computing device as the claim generator (e.g. Qualcomm's trusted location firmware)
`remoteProvider`	Assertion contents came from a remote data source controlled by the signer or claim generator vendor (e.g. BBC asset repository server)
`remoteProvider.external`	Assertion contents came from an external, remote data source that is not the signer or claim generator vendor (e.g. Google geolocation API service)
`humanEntry.anonymous`	Assertion contents were entered by a human that wishes to remain anonymous
`humanEntry.credentialed`	Assertion contents were entered by a human who has a Verifiable Credential. If this is the value of `type`, then the `credential` field must also be declared under `source`.

The `details` field is a human-readable string that provides additional information about the dataSource, e.g. the name of the API used to provide the assertion contents, or the URL of the server from which the contents were

provided. For example, a broad location assertion source may have a `type` value of `remoteProvider.3rdParty`, with the `details` value set to `www.googleapis.com/geolocation/v1/geolocate`.

If the value of the `type` field is `humanEntry.credentialed`, then an `actors` field is also required. The `actors` field shall contain an array or one or more actor objects, each one which shall have a `credentials` field containing at least one `hashed_uri` to a W3C Verifiable Credentials associated with the people or organizations who are responsible for the assertion contents.

17.3.1.2. The `reviewRatings`

When present, the `reviewRatings` array provides a place for the claim generator to provide one or more `rating` objects on the quality (or lack thereof) of an assertion. A `reviewRatings` shall not be present if a `dataSource` object is present with a `type` field whose value is either `humanEntry.anonymous` or `humanEntry.credentialed`.

The `value` field of the `rating` object shall be present with any integer value from *1* (worst) through *5* (best). If present, the `explanation` field shall contain a human-consumable string description of the type of rating. In addition, an optional machine-readable `code` field which defines assertion-specific evaluation outcome codes may be provided. The value of the `code` field is assembled with labels in the format described in Section 5.2, "Labels". The value can be one of the following specification-defined values, or entity-specific labels can be used as an extension mechanism.

Value of `code`	**Applicable Assertion**	**Meaning**
`actions.unknownActionsPerformed`	`c2pa.action`	The `actions` assertion does not contain a full list of all actions performed in the authoring tool (e.g. because of the use of a 3rd party filter whose effect is unknown to the authoring tool)
`depthMap.sceneMismatch`	`c2pa.depthmap.GDepth`	The contents of the depth map assertion do not correspond to the scene portrayed in the primary presentation in the asset (e.g. because of a picture-of-picture attack)
`thumbnail.primaryMismatch`	`c2pa.claim.thumbnail`	The thumbnail's contents do not match the contents of the primary presentation in the asset.
`stds.schema-org.CreativeWork.misattributed`	`stds.schema-org.CreativeWork`	One or more of the roles listed in a `CreativeWork` assertion is misattributed to the wrong actor (e.g. the wrong credit is given for the `editor` role)
`stds.schema-org.CreativeWork.missingAttribution`	`stds.schema-org.CreativeWork`	An attribution for a role in a `CreativeWork` assertion is missing (e.g. a person who played the role of `producer` isn't credited)
`stds.iptc.location.inaccurate`	`stds.iptc`	The reported location is inaccurate (e.g. reported to be in New York, NY but appears to be in another city entirely)

17.3.1.3. References

Because the `reference` field of the `assertion metadata` assertion is a standard `hashed_uri`, it is also possible to have an `assertion metadata` assertion refer to assertions in other manifests than the active one. For example, the active manifest could include an `assertion metadata` assertion that validates the `precise location` assertion present in an ingredient's manifest.

17.3.1.4. DateTime

If a `dateTime` field is present, its value shall be a date time string that complies with ISO 8601.

17.4. Standard C2PA Assertion Summary

The standard C2PA assertions are:

Type	Assertion	Schema	Serialization
Data Hash	c2pa.hash.data	C2PA	CBOR
BMFF-based Hash	c2pa.hash.bmff	C2PA	CBOR
Soft Binding	c2pa.soft-binding	C2PA	CBOR
Cloud Data	c2pa.cloud-data	C2PA	CBOR
Thumbnail	c2pa.thumbnail.claim (claim creation time) c2pa.thumbnail.ingredient (importing an ingredient)	C2PA	CBOR
Actions	c2pa.actions	C2PA	CBOR
Ingredient	c2pa.ingredient	C2PA	CBOR
GDepth Depthmap	c2pa.depthmap.GDepth	https://developers.google.com/depthmap-metadata/reference	CBOR
Exif information	stds.exif	C2PA	JSON-LD
IPTC Photo Metadata	stds.iptc.photo-metadata	C2PA	JSON-LD
Claim Review	stds.schema-org.ClaimReview	Schema.org ClaimReview	JSON-LD
Creative Work	stds.schema-org.CreativeWork	Schema.org CreativeWork	JSON-LD

Type	Assertion	Schema	Serialization
Assertion Metadata	c2pa.assertion.metadata	C2PA	CBOR

17.5. Data Hash

Portion(s) of a non-BMFF-based asset that a claim generator wishes to uniquely identify with a hard binding (aka cryptographic hash) shall be described using data hash assertions. The types of hashes which can be created and stored in such an assertion are described in Section 12.1, "Hashing" and the value shall be present in the `hash` field.

Each data hash assertion defines a specified range of bytes over which the hash has been computed. By default, this range is the entire asset containing this assertion or the entire asset referred to by the `url` value in the assertion. If only a portion of the asset shall be hashed, then the range(s) to be excluded shall be present in the array value of the `exclusions` field.

NOTE The `url` value provides flexibility for hashing assets that may be represented in multiple chunks or portions, local or remote.

A Data Hash assertion shall have a label of `c2pa.hash.data`.

17.5.1. Schema and Example

The CDDL Definition for this type is:

```
; Also check optionality within the hash-map
; The data structure used to store the cryptographic hash of some or all of the asset's data

; and additional information required to compute the hash.
data-hash-map = {
  ? "exclusions": [1* EXCLUSION_RANGE-map],
  ? "alg":tstr .size (1..max-tstr-length), ; A string identifying the cryptographic hash
algorithm used to compute the hash in this assertion, taken from the C2PA hash algorithm
identifier list. If this field is absent, the hash algorithm is taken the `alg` value of the
enclosing structure. If both are present, the field in this structure is used. If no value
is present in any of these places, this structure is invalid; there is no default.
  "hash": bstr, ; byte string of the hash value
  "pad": bstr, ; byte string used for filling up space
  ? "name": tstr .size (1..max-tstr-length), ; (optional) a human-readable description of
what this hash cover
  ? "url": uri, ; (optional) a file or http(s) URL to where the bytes that are being hashed
can be found.
              ; This is useful for cases where the data lives in a different file chunk or
side-car than the claim.
}

EXCLUSION_RANGE-map = {
  "start": int, ; Starting byte of the range
  "length": int, ; Number of bytes of data to exclude
}
```

An example in CBOR-Diag is shown below:

```
{
  "exclusions": [
    {
      "start": 9960,
      "length": 4213
    }
  ],
  "alg" : "sha256",
  "hash": 'Auxjtmax46cC2N3Y9aFmBO9Jfay8LEwJWzBUtZ0sUM8gA=',
  "pad" : 'abc',
  "name": "JUMBF manifest"
}
```

Normally, the `start` and `length` values of an `exclusion` shall be written in their preferred serialization (i.e., "as short as possible"). However, when a data hash assertion needs to be created but the `start` and `length` values are not yet known, they shall be created "as large as possible", which would be as a 32-bit integer.

The `pad` value shall always be present but shall be a byte string of length 0 unless used to replace (aka "pad") bytes during multiple pass processing.

NOTE Section 9.4, “Multiple Step Processing” describes how to fill in the correct values and adjust the padding.

17.6. BMFF-Based Hash

Portion(s) of a BMFF-based asset that a claim generator wishes to uniquely identify with a hard binding (aka cryptographic hash) shall be described using BMFF-based Hash assertions.

A BMFF-based Hash assertion shall have a label of `c2pa.hash.bmff`.

To create the hash specified in the value field of a bmff hash, all bytes of the file are added to the hash excluding those BMFF boxes which match any exclusion entry in the exclusions array. Any box included in the hash also includes its box headers; any box excluded from the hash also excludes its box headers.

A box matches an exclusion entry in the exclusions array if and only if all of the following conditions are met.

- The box’s location in the file exactly matches the exclusion entry’s xpath field.
- If length is specified in the exclusion entry, the box’s length exactly matches the exclusion entry’s length field. Note: The length includes the box headers.
- If version is specified in the exclusion entry, the box is a FullBox and the box’s version exactly matches the exclusion entry’s version field.
- If flags (byte array of exactly 3 bytes) is specified in the exclusion entry, the box is a FullBox and the box’s flags (bit(24), i.e., 3 bytes) exactly matches the exclusion entry’s flags field.

- If data (array of objects) is specified in the exclusion entry, then for each item in the array, the box's binary data at that item's offset field exactly matches that item's bytes field.

The xpath syntax shall be limited to the following strict subset.

- Only abbreviated syntax shall be used.
- Only full paths shall be used.
- Only node selection via `node` or `node[integer]` shall be used.
- Descendent syntax, i.e., `//`, shall NOT be used.
- All nodes shall be BMFF `4cc` codes.

Complete Syntax:

```
  xpath = '/' nodes
  nodes = node
        | node '/' nodes
  node = box4cc
       | box4cc '[' integer ']'
Where:
  box4cc is any 4cc allowed by ISO/IEC 14496-12 for a BMFF box.
  integer is any non-zero positive integer with no leading zeros.
```

A non-leaf xpath node shall only point to a container box that has no fields of its own (i.e. contains no data, only child boxes) and that does not inherit from FullBox. This ensures that a c2pa verifier does not need to be aware of the syntax and semantics of unusual boxes that contain other boxes. If a child box of such an unusual box needs to be excluded in full or in part, the exclusion's xpath shall point to the unusual box itself and the subset member shall exclude the byte ranges containing the excluded child box data. For example, the `sgpd` box contains other boxes but is unusual in that it inherits from FullBox; as such, if excluding child box(es) from `sgpd` is required, the assertion shall use an xpath pointing to the `sgpd` itself (e.g. `/moof/traf/sgpd`) and shall use the subset member to exclude the desired bytes.

If the manifest is embedded into the file, the box containing it shall be one of the entries in the exclusions array and all file offsets shall be excluded wherever they appear. Refer to Section 10.2.2, "Embedding manifests into BMFF-based assets" for more information.

Because the box that contains the manifest is at the root and always excludes file offsets, the hash can be computed before the manifest is created and the manifest can then be added without changing any existing portion of the file or invalidating the hash.

If a non-root excluded box is removed after the manifest is created, it shall be replaced with a `free` box of the same size to ensure that the hash of its parent box is not invalidated due to its size changing. If it is expected that a non-root excluded box may be added after the manifest is created, then, at manifest creation time, a `free` box shall be inserted with sufficient space for the excluded box and that `free` box shall also be excluded by an exclusion entry using its full xpath. When the excluded box is added, the `free` box shall be shrunk (or removed) to make space for the added box.

17.6.1. Schema and Example

The CDDL Definition for this type is:

```
bmff-hash-map = {
  "exclusions": [1* exclusions-map],
  ? "alg": tstr .size (1..max-tstr-length), ; A string identifying the cryptographic hash
algorithm used to compute this hash, taken from the C2PA hash algorithm identifier list. If
this field is absent, the hash algorithm is taken from an enclosing structure as defined by
that structure. If both are present, the field in this structure is used. If no value is
present in any of these places, this structure is invalid; there is no default.
  ? "hash": bstr, ; For non-fragmented mp4, this shall be the hash of the entire BMFF file
excluding boxes listed in the exclusions array; if the merkle field is present, this hash
also excludes all 'mdat' boxes.  For fragmented mp4, this field is required to be absent.
  ? "merkle": [1* merkle-map], ; A set of merkle tree rows and the associated data required
to enable verification of a single 'mdat' box, multiple 'mdat' boxes, and/or individual
fragment files within the asset.
  ? "name": tstr .size (1..max-tstr-length), ; optional) a human-readable description of
what this hash covers.
  ? "url": uri, ; (optional) a file or http(s) URL to where the bytes that are being hashed
lived.  This is useful for cases where the data lives in a different file chunk or side-car
than the claim.
}

;(optional) CBOR byte string of exactly 3 bytes.
flag-type = bytes

flag-t = flag-type .eq 3

exclusions-map = {
  "xpath": tstr, ; Location of box(es) to exclude from the hash starting from the root node
as an xpath formatted string of version https://www.w3.org/TR/xpath-10/ with highly
constrained syntax.
  ? "length": int, ; (optional) Length that a leafmost box must have to exclude from the
hash.
  ? "data": [1* data-map], ; (optional) The data in the leafmost box at the specified offset
must be identical to the specified data for the box to be excluded from the hash.
  ? "subset":[1* subset-map], ; (optional) Only this portion of the excluded box shall be
excluded from the hash.  Each entry in the array must have a monotonically increasing
offset.  No subset within the array may overlap.
  ? "version": int, ; (optional) Version that must be set in a leafmost box for the box to
be excluded from the hash.  Shall only be specified for a box that inherits from FullBox.
  ? "flags": flag-t,  ; (optional) byte string of exactly 3 bytes.  The 24-bit flags that
must be set in a leafmost box for the box to be excluded from the hash.  Shall only be
specified for a box that inherits from FullBox.
}

data-map = {
  "offset": int,
  "value" : bstr,
}
subset-map = {
  "offset": int,
  "length": int,
}

; Each entry in a map is a merkle tree rows and the associated data required to enable
validation of a single
; 'mdat' box or multiple 'mdat' boxes within the asset.",
merkle-map = {
  "uniqueId": int, ; 1-based unique id used to differentiate across files to determine which
```

```
merkle tree should be used to verify a given 'mdat' box.
  "localId": int, ; Local id used to differentiate across multiple 'mdat' boxes within a
single file to determine which merkle tree should be used to verify that 'mdat' box.
  "count": int, ; Number of leaf nodes in the Merkle tree.  Null nodes are not included in
this count.
  ? "alg": tstr .size (1..max-tstr-length), ; A string identifying the cryptographic hash
algorithm used to compute the hashes in this merkle tree, taken from the C2PA hash algorithm
identifier list. If this field is absent, the hash algorithm is taken the `alg` value of the
enclosing structure as defined by that structure. If both are present, the field in this
structure is used. If no value is present in any of these places, this structure is invalid;
there is no default.
  ? "initHash": bstr, ; For fragmented mp4, this field is required to be present and shall
be the hash of the initialization segment for chunks hashed by this merkle tree.  For non-
fragmented mp4, this field is required to be absent.
  "hashes": [1* bstr], ; An ordered array representing a single row of the Merkle tree which
may be the leaf-most row, root row, or any intermediate row.  The depth of the row is
implied by (shall be computed from) the number of items in this array.
}
```

An example in CBOR-Diag for a monolithic mp4 file asset where the `mdat` box is verified as a unit is shown below:

```
{
  "exclusions": [
    {
      "xpath": "/moov[1]/pssh"
    },
    {
      "xpath": "/emsg",
      "data": [
        {
          "offset": 20,
          "value": b64'r3avWCpXHkmKHATFsV0Q5g'
        }
      ]
    }
  ],
  "name": "Example BMFF-hash object",
  "hash": b64'EiAuxjtmax46cC2N3Y9aFmBO9Jfay8LEwJWzBUtZ0sUM8gA'
}
```

An example in CBOR-Diag for an asset composed of fragmented mp4 files is shown below:

```
{
  "exclusions": [
    {
      "xpath": "/c2pa"
    },
    {
      "xpath": "/moov[1]/pssh"
    },
    {
      "xpath": "/emsg",
      "length": 200,
      "data": [
        {
          "offset": 5,
          "value": b64'9Q=='
        },
```

```
        {
          "offset": 20,
          "value": b64'UAJXD79SlkG9rfnmcsqTUA=='
        },
        {
          "offset": 70,
          "value": b64'OxKM'
        }
      ],
      "subset": [
        {
          "offset": 5,
          "length": 7
        },
        {
          "offset": 20,
          "length": 28
        },
        {
          "offset": 45,
          "length": 63
        },
        {
          "offset": 80,
          "length": 112
        }
      ],
      "version": 1,
      "flags": b64'ZDNx'
    }
  ],
  "alg": "sha256",
  "merkle": [
    {
      "uniqueId": 17,
      "localId": 19,
      "count": 23,
      "initHash": b64'Hf0IgeqbL0m+FTTLpUWwsDGR8pvhUR1AlwvaXjQ0qGY='
      "hashes": [ b64'HvWZOxKMfkSatRAygs8DJfnEEcN/G1BNi359NdIDxbQ=', b64'HvWZOxKMfkSatRAygs
8DJfnEEcN/G1BNi359NdIDxbQ=' ]
    },
    {
      "uniqueId": 34,
      "localId": 38,
      "count": 69,
      "initHash": b64'Hf0IgeqbL0m+FTTLpUWwsDGR8pvhUR1AlwvaXjQ0qGY=',
      "hashes": [ b64'9Zk7Eox+RJq1EDKCzwMl+cQRw38bUE2Lfn010gPFtB0=', b64'9Zk7Eox+RJq
1EDKCzwMl+cQRw38bUE2Lfn010gPFtB0=', b64'mTsSjH5EmrUQMoLPAyX5xBHDfxtQTYt+fTXSA8W0Hf0=', b
64'mTsSjH5EmrUQMoLPAyX5xBHDfxtQTYt+fTXSA8W0Hf0=', b64'OxKMfkSatRAygs8DJfnEEcN/G1BNi
359NdIDxbQd/Qg=' ]
    },
    {
      "uniqueId": 51,
      "localId": 57,
      "count": 46,
      "initHash": b64'Hf0IgeqbL0m+FTTLpUWwsDGR8pvhUR1AlwvaXjQ0qGY=',
      "hashes": [ b64'OxKMfkSatRAygs8DJfnEEcN/G1BNi359NdIDxbQd/Qg=' ]
    }
  ],
  "name": "Example BMFF-hash object for fMP4"
}
```

A pseudo-code implementation of this algorithm follows.

```
offset = 0
While (offset < length of file)
    Starting at offset, locate the first byte of the first box that matches any entry in the
exclusions array, call this first_excluded_byte
        If no such box is found, set first_excluded_byte = length of file
    Determine the length of that box, call this excluded_byte_count
        If no such box was found, set excluded_byte_count = 0
    To the hash, add all bytes between offset and first_excluded_byte minus one (inclusive)
    If first_excluded_byte < length of file and there exists a subset array within the
exclusion that determined the value of first_excluded_byte
        set next_included_begin = first_excluded_byte
        For each entry in the subset array within the exclusion that determined the value of
first_excluded_byte
            Set next_excluded_begin = this subset array entry's offset field plus
first_excluded_byte
            If next_excluded_begin > next_included_begin
                To the hash, add all bytes between next_included_begin and
next_excluded_begin minus one (inclusive)
            Set next_included_begin  = this subset array entry's length field plus
next_excluded_begin
        If next_included_begin < first_excluded_byte + excluded_byte_count
            To the hash, add all bytes between next_included_begin and first_excluded_byte +
excluded_byte_count minus one (inclusive)
    Set offset = first_excluded_byte + excluded_byte_count
```

17.6.2. Exclusion list profiles

NOTE

Editor's Note

A set of recommended exclusions for various types of media would go here…

17.7. Soft Binding

If a claim generator wishes to provide a soft binding for the asset's content, it shall be described using a soft binding assertion. The types of soft bindings which can be created and stored in such an assertion are described in Section 8.3, "Soft Bindings".

NOTE

The `url` value provides flexibility for hashing assets that may be represented in multiple chunks or portions, local or remote.

A Soft Binding assertion shall have a label of `c2pa.soft-binding`.

17.7.1. Schema and Example

The CDDL Definition for this type is:

```
;The data structure used to store one or more soft bindings across some or all of the
asset's content
soft-binding-map = {
  "alg": tstr, ; A string identifying the soft binding algorithm and version of that
algorithm used
              ; to compute the value, taken from the C2PA soft binding algorithm identifier
list. If this field is absent, the algorithm is taken from the `softbinding-alg` value of
the enclosing structure. If both are present, the field in this structure is used.
              ; If no value is present in any of these places, this structure is invalid;
there is no default.
  "blocks": [1* soft-binding-block-map],
  "pad": bytes, ; byte string used for filling up space
  ? "name": tstr .size (1..max-tstr-length), ; (optional) a human-readable description of
what this hash covers
  ? "alg-params": bstr, ; (optional) CBOR byte string describing parameters of the soft
binding algorithm.
                        ; If this field is absent, the algorithm is taken from the
`softbinding-alg-params`
                        ; value of the enclosing structure, if present."
  ? "url": uri, ; (optional) a file or http(s) URL to where the bytes that are being hashed
lived.
                ; This is useful for cases where the data lives in a different file chunk or
side-car
                ; than the claim.
}

soft-binding-block-map = {
  "scope": soft-binding-scope-map,
  "value": bstr, ; CBOR byte string describing, in algorithm specific format,
                ; the value of the soft binding computed over this block of digital content"
}

soft-binding-scope-map = {
  ? "extent": bstr, ;CBOR byte string describing, in algorithm specific format,
                    ; the part of the digital content over which the soft binding value has
been computed"
  ? "timespan":soft-binding-timespan-map,
}

soft-binding-timespan-map = {
  "start": time, ; Start of the time range over which the soft binding value has been
computed.
  "end": time,   ; End of the time range over which the soft binding value has been
computed.
}
```

An example in CBOR-Diag is shown below:

```
{
  "url": 32("http://example.c2pa.org/media.mp4"),
  "alg": "phash",
  "blocks": [
    {
      "scope": {
        "extent": b64'c2NvcGUxCg==',
        "timespan": {
          "start": 1(0),
          "end": 1(4006)
        }
      },
      "value": b64'dmFsdWUxCg=='
    },
    {
      "scope": {
        "extent": b64'c2NvcGUyCg==',
        "timespan": {
          "start": 1(4007),
          "end": 1(7359)
        }
      },
      "value": b64'dmFsdWUyCg=='
    }
  ],
  "pad": h'00'
}
```

The soft binding algorithm used shall be present as the value of the `alg` field, and the blocks over which is was applied shall be listed in the `blocks` field. If the algorithm used requires any additional parameters, they should be present as the value of `alg-params`.

17.8. Cloud Data

There are use cases where storing the data for the assertion remotely, such as in the cloud, is better than embedded inside the asset, especially when the data is large. For any such cases, it is possible to use a special type of assertion that serves as a reference to that information.

A Cloud Data assertion shall have a label of `c2pa.cloud-data`.

17.8.1. Schema and Example

The CDDL Definition for this type is:

```
; Assertion that references the actual assertion stored in the cloud
cloud-data-map = {
  "label": tstr, ; label for the cloud-based assertion (eg.c2pa.claim.date)
  "size": size-type, ; Number of bytes of data
  "location": $hashed-uri-map, ; a file or http(s) URL to where the bytes that are being
hashed can be found
  "content_type": tstr, ; media/MIME type for the data
  ? "metadata": $assertion-metadata-map,  ; additional information about the assertion
}

; size is minimum 1 in multiples of 1.0
size-type = int .ge 1
```

An example in CBOR-Diag is shown below:

```
{
  "label": "c2pa.thumbnail.claim.jpg",
  "size": 98765,
  "location": {
    "url": "https://some.storage.us/foo",
    "hash": b64'zP84FPSremIrAQHlhw+hRYQdZp/+KggnD0W8opXlIQQ='
  },
  "content_type": "application/jpeg"
}
```

17.9. Thumbnail

A **thumbnail** assertion provides an approximate visual representation of the asset at a specific event in the lifecycle of an asset. There are currently two specific events - ingredient import and claim creation - each using a unique label for the assertion.

For thumbnails created at claim creation time, the Thumbnail assertion shall have a label that starts with `c2pa.thumbnail.claim` and be followed by the IANA registry image type (e.g., `c2pa.thumbnail.claim.png`). For each of these types of thumbnails, there can be only one per claim.

When importing an ingredient (see Section 9.3.2.2, “Adding Ingredients”), it is preferable to reference that ingredient’s own manifest-stored thumbnail. However, some ingredients may not include a thumbnail assertion, or even a manifest. In that case, a new thumbnail of the ingredient should be generated, and a new thumbnail assertion in the active manifest created. The Thumbnail assertion shall have a label that starts with `c2pa.thumbnail.ingredient` and be followed by an underscore (_ (U+005F)) then a unique ID such as a simple monotonically increasing integer and ending with the image type. For example, an ingredient thumbnail of type `jpeg` could have label `c2pa.thumbnail.ingredient_1.jpg`.

The data in a thumbnail assertion is the bits of a file (such as a raster image) in whatever format is desired by the claim generator. The Embedded File content type (ISO 19566-5:AMD-1), `bfdb`, shall be used to contain the thumbnail’s data.

17.10. Actions

An **actions** assertion provides information on edits and other actions taken that affect the asset's content. There will be an array of actions - each action declaring *what* took place on the asset, *when* it took place, along with possible other information such as what software performed the action.

An Actions assertion shall have a label of `c2pa.actions`.

Actions are modeled after XMP ResourceEvents.

Every action present in the array shall declare the date and time when the action was performed as an ISO 8601-compliant string as the value of the `when` field. The value of the `action` field shall be present and shall be either a well-known common action name (`c2pa.resized`, `c2pa.edited`, etc.) or entity-specific action name (`adobe.ps.gaussian_blur`, etc.).

The set of defined names, prefixed with `c2pa.` are:

Action	Meaning	Source
c2pa.converted		XMP
c2pa.copied		XMP
c2pa.created		XMP
c2pa.cropped		XMP
c2pa.edited		XMP
c2pa.filtered		XMP
c2pa.formatted		XMP
c2pa.version_updated		XMP
c2pa.printed		XMP
c2pa.published		XMP
c2pa.managed		XMP
c2pa.produced		XMP
c2pa.resized		XMP
c2pa.saved		XMP
c2pa.transcoded	A direct conversion of one encoding to another, including resolution scaling, bitrate adjustment and encoding format change. Does not include any adjustments that would affect the "editorial" meaning of the content.	C2PA

Action	Meaning	Source
c2pa.repackaged	A conversion of one packaging or container format to another. Content may be repackaged without transcoding. Does not include any adjustments that would affect the "editorial" meaning of the content.	C2PA
c2pa.placed	Added/Placed an ingredient into the asset	C2PA
c2pa.unknown	Something happened, but the claim_generator can't specify what	C2PA

In addition, an action may include a `actors` key which can point to the identity of one or more human actor who performed the action. The value of the `actors` key is an object with a single key: `credentials`. The `credentials` key's value is an array of `hashed_uri`, which can point to a W3C Verifiable Credential in the Credential Store.

When using a `c2pa.transcoded`, `c2pa.repackaged`, or a `c2pa.placed` action, the `parameters` field shall contain the JUMBF URI to the related ingredient assertion.

17.10.1. Asset Renditions

Asset renditions are a common occurrence when distributing media on the internet. These renditions are often created for the purpose of delivering media to consumers in differing connectivity, screen resolution, and other environments. We can use the `actions` assertion to help consuming actors understand the intention of certain claim creators to create asset renditions.

The presence of only the `c2pa.transcoded` and/or `c2pa.repackaged` actions in a `c2pa.actions` assertion provides a signal to the manifest consumer that the signer is asserting that no "editorial" changes have happened between the ingredient asset(s) and this one. Editorial changes are those that alter the intent and/or meaning of the content.

The additional presence of a single "parentOf" ingredient provides a further signal to the manifest consumer that the signer is asserting that the asset has been derived directly from that parent.

17.10.2. Schema and Example

The CDDL Definition for this type is:

```
actions-map = {
  "actions" : [1* action-items-map],  ; list of actions
  ? "metadata": $assertion-metadata-map, ; additional information about the assertion
}


buuid = #6.37(bstr)

$action-choice /= "c2pa.converted"
$action-choice /= "c2pa.copied"
$action-choice /= "c2pa.created"
$action-choice /= "c2pa.cropped"
$action-choice /= "c2pa.edited"
$action-choice /= "c2pa.filtered"
$action-choice /= "c2pa.formatted"
$action-choice /= "c2pa.version_updated"
$action-choice /= "c2pa.printed"
$action-choice /= "c2pa.published"
$action-choice /= "c2pa.managed"
$action-choice /= "c2pa.produced"
$action-choice /= "c2pa.resized"
$action-choice /= "c2pa.saved"
$action-choice /= "c2pa.transcoded"
$action-choice /= "c2pa.repackaged"
$action-choice /= "c2pa.placed"

actor-map = {
  ? "credentials": [1* $hashed-uri-map]
}

action-items-map = {
  "action": $action-choice,
  "when": tdate, ; Timestamp of when the action occurred.
  ? "softwareAgent": tstr .size (1..max-tstr-length), ;The software agent that performed the
action.
  ? "changed": tstr .size (1..max-tstr-length), ; A semicolon-delimited list of the parts of
the resource that were changed since the previous event history. If not present, presumed to
be undefined. When tracking changes and the scope of the changed components is unknown, it
should be assumed that anything might have changed.
  ? "instanceID": buuid, ; The value of the xmpMM:InstanceID property for the modified
(output) resource
  ? "parameters": tstr .size (1..max-tstr-length), ; Additional description of the action
  ? "actors": [1* actor-map], ; An array of the creators that undertook this action
}
```

An example in CBOR-Diag is shown below:

```
{
  "actions": [
    {
      "action": "c2pa.filtered",
      "when": 0("2020-02-11T09:00:00Z"),
      "softwareAgent": "Adobe Photoshop 2020 for Windows",
      "changed": "change1,change2",
      "instanceID": 37(h'ed610ae51f604002be3dbf0c589a2f1f'),
      "actor": [
        {
          "credential": [
            {
              "url": "self#jumbf=c2pa/acme:urn:uuid:F9168C5E-CEB2-4faa-B6BF-
329BF39FA1E4/c2pa.credentials/Joe_Bloggs",
              "alg": "sha256",
              "hash": b64'hoOspQQ1lFTy/4Tp8Epx670E5QW5NwkNR+2b30KFXug='
            }
          ]
        }
      ]
    }
  ],
  "metadata": {
    "reviewRating": [
      {
        "value": 1,
        "explanation": "Content bindings did not validate"
      }
    ],
    "dateTime": 0("2021-06-28T16:34:11.457Z")
  }
}
```

17.11. Ingredient

When assets are composed together, for example placing an image into a layer in Photoshop or an audio clip into a video in Premiere, it is important that information about any claim from the placed asset be recorded into the new asset to provide a way to understand the entire history of the new composed asset. This is also true when an existing asset is used to create a derived asset or asset rendition.

An Ingredient assertion shall have a label of `c2pa.ingredient`.

NOTE Since there will most likely be more than one ingredient assertion, the use of the monotonically increasing index in the label would be used (e.g., `c2pa.ingredient__1`, `c2pa.ingredient__2`).

This is modelled on the XMP Ingredient and Pantry model, as described in the Partner Guide to XMP for Dynamic Media and Asset Relationships in XMP. That model relies on the fact that each asset used in the construction of a document has, at the time of inclusion, at least one unique identifier.

If the ingredient being added contains XMP, then asset's `xmpMM:DocumentID` becomes the `documentID` field while the `xmpMM:InstanceID` becomes the `instanceID`. However, if the ingredient being added does not have

any associated XMP, then it may be possible for the XMP to be created and added to the ingredient itself and the identifiers used as described above.

When it is not possible or desirable to create XMP for an ingredient, then some other unique identifier for the ingredient shall be used instead for the required values. In this situation, the `instanceID` field shall contain the unique identifier and the `documentID` field shall not be present.

When adding an ingredient, its relationship to the current asset shall be described. These are the possible values of the relationship field and their meanings:

Value	Meaning
`parentOf`	The current asset is a derived asset or asset rendition of this ingredient.
`componentOf`	The current asset is composed of multiple parts, this ingredient being one of them.

The value of `dc:title` shall be a human-readable name for the ingredient, which may be taken either from the asset's XMP or the asset's name in a local or remote (e.g., cloud-based) filesystem. The Media Type of the ingredient shall be declared in `dc:format`.

If the ingredient has an existing C2PA Manifest, then that manifest shall be copied into the C2PA Manifest Store for the asset. The URI reference to that C2PA Manifest shall be stored as the value of `c2pa_manifest`.

It is recommended that, if it is technically able to, the processor validate the content hard bindings (i.e., hashes) provided in the claim and provide a `review` of them as part of the assertion metadata. When present, the assertion metadata shall contain at least one entry in the `reviewRatings` array, whose `value` field has a value which is either *1* (did not validate) or *5* (successfully validated). No other values are permitted for the `value` in this use of reviews.

17.11.1. Schema and Example

The CDDL Definition for this type is:

```
;Assertion that describes an ingredient used in the asset
ingredient-map = {
  "dc:title": tstr, ; name of the ingredient
  "dc:format": format-string, ; Media Type of the ingredient
  "documentID": tstr, ; value of the ingredient's `xmpMM:DocumentID`
  "instanceID": tstr, ; unique identifier, such as the value of the ingredient's
`xmpMM:InstanceID`
  "relationship": $relation-choice, ; The relationship of this ingredient to the asset it is
an ingredient of.
                                    ; For example, if an ingredient with a 'parentOf'
relationship is added to
                                    ; an asset, then the asserter is stating that the
current asset is a derived asset of the ingredient.
  ? "c2pa_manifest": $hashed-uri-map, ; hashed_uri reference to the C2PA Manifest of the
ingredient
  ? "thumbnail": $hashed-uri-map, ; hashed_uri reference to an ingredient thumbnail
  "metadata": $assertion-metadata-map ; additional information about the assertion
}

format-string = tstr .regexp "^\\w+\/[-+.\\w]+$"
; Assertion that describes an ingredient used in the asset
$relation-choice /= "parentOf"
$relation-choice /= "componentOf"
```

An example in CBOR-Diag is shown below:

```
{
  "dc:title": "image 1.jpg",
  "dc:format": "image/jpeg",
  "documentID" : "uuid:87d51599-286e-43b2-9478-88c79f49c347",
  "instanceID" : "uuid:7b57930e-2f23-47fc-affe-0400d70b738d",
  "c2pa_manifest" : {
      "url": "self#jumbf=c2pa/urn:uuid:5E7B01FC-4932-4BAB-AB32-D4F12A8AA322",
      "hash": b64'1kjJTO108b71cL95UxgfHD3eDgk9VrCedW8n3fYTRMk='
  },
  "thumbnail" : {
      "url": "self#jumbf=cp2a/acme:urn:uuid:F9168C5E-CEB2-4faa-B6BF-
329BF39FA1E4/c2pa.thumbnail.ingredient_1.jpg",
      "hash": b64'UjRAYWiAq4lfCRDmksWAlDJN/XtHHFFwMWymsZsm3j8='
  },
  "relationship": "parentOf",
  "metadata": {
    "reviewRating": [
      {
        "value": 5,
        "explanation": "Content bindings validated"
      }
    ],
    "dateTime": 0("2021-06-28T16:49:32.874Z")
  }
}
```

When adding an ingredient, it may be useful to also include a thumbnail of the ingredient to help establish the state of the ingredient at the time of import. For that purpose, an ingredient import thumbnail should be added as described in [_thumbnails] and referenced herein via a URI reference.

17.12. Depthmap

A Depthmap assertion provides a 3D description of the scene being captured by a camera. A Depthmap assertion may contain a pre-computed depth map, or data which can later be used to compute a depth map by downstream ingestion or viewing software (e.g., left/right stereo images).

All Depthmap assertions shall have a label that starts with `c2pa.depthmap` and be followed by a third section that identifies the type of depth map.

C2PA Depthmap assertions shall be captured optically, not inferred from a single 2D image via, for example, a machine learning model.

17.12.1. GDepth Depthmap

A GDepth depth map assertion leverages the well-established GDepth format to encode a pre-computed depth map.

A GDepth Depthmap assertion shall have a label of `c2pa.depthmap.GDepth`.

The schema for the data stored in this assertion should always mirror the schema at https://developers.google.com/depthmap-metadata/reference.

NOTE	There is no need to worry about splitting up the GDepth data when it grows beyond 64KB, as that limit existed in XMP to accommodate APP1 segment size limitations.

17.12.2. Schema and Example

The CDDL Definition for this type is:

```
; Assertion that encodes a GDepth-formatted 3D depth map of the captured scene
depthmap-gdepth-map = {
  "GDepth:Format": format-type, ; The format that describes how to convert the depthmap data
into a valid float-point depthmap. Current valid values are 'RangeInverse' and 'RangeLinear'
  "GDepth:Near": float, ; The near value of the depthmap in depth units
  "GDepth:Far": float,  ; The far value of the depthmap in depth units
  "GDepth:Mime": mime-type,  ;  The mime type for the base64 string describing the depth
image content, e.g. 'image/jpeg' or 'image/png'",
  "GDepth:Data": base64-string-type,  ; The base64 encoded depth image. Please see GDepth
encoding page at developers.google.com. The depthmap will be stretched-to-fit the
corresponding color image
  ? "GDepth:Units": unit-type,   ; The units of the depthmap, e.g. 'm' for meters or 'mm'
for millimeters
  ? "GDepth:MeasureType": depth-meas-type,  ; The type of depth measurement. Current valid
values are 'OpticalAxis' and 'OpticRay
  ? "GDepth:ConfidenceMime": confidence-mime-type,  ; The mime type for the base64 string
describing the confidence image content, e.g. 'image/png'.",
  ? "GDepth:Confidence": base64-string-type,   ; The base64 encoded confidence image. Please
see GDepth encoding page at developers.google.com. The confidence map should have the same
size as the depthmap
  ? "GDepth:Manufacturer": tstr .size (1..max-tstr-length),  ; The manufacturer of the
device that created this depthmap
  ? "GDepth:Model": tstr .size (1..max-tstr-length),  ; The model of the device that created
this depthmap
  ? "GDepth:Software": tstr .size (1..max-tstr-length), ; The software that created this
depthmap
  ? "GDepth:ImageWidth": float, ; The width in pixels of the original color image associated
to this depthmap. This is NOT the depthmap width. If present, apps must update this property
when scaling, cropping or rotating the color image. Clients use this property to verify the
integrity of the depthmap w.r.t. the color image
  ? "GDepth:ImageHeight": float, ; The height in pixels of the original color image
associated to this depthmap. This is NOT the depthmap height. If present, apps must update
this property when scaling, cropping or rotating the color image. Clients use this property
to verify the integrity of the depthmap w.r.t. the color image
  ? "metadata": $assertion-metadata-map, ; additional information about the assertion
}

base64-string-type = tstr

$mime-choice /= "image/jpeg"
$mime-choice /= "image/png"

mime-type = $mime-choice .default "image/jpeg"
confidence-mime-type = $mime-choice .default "image/png"

$format-choice /= "RangeInverse"
$format-choice /= "RangeLinear"

format-type = $format-choice .default "RangeInverse"

; Unit can be meter represented as "m" or could be millimeter represented as "mm"
$unit-choice /= "m"
$unit-choice /= "mm"
unit-type = $unit-choice .default "m"

$depth-meas-choice /= "OpticalAxis"
$depth-meas-choice /= "OpticRay"
depth-meas-type = $depth-meas-choice .default "OpticalAxis"
```

An example in CBOR-Diag is shown below:

```
{
  "GDepth:Format": "RangeInverse",
  "GDepth:Near": 29.3,
  "GDepth:Far": 878.7,
  "GDepth:Mime": "image/jpeg",
  "GDepth:Data": "hoOspQQ1lFTy/4Tp8Epx670E5QW5NwkNR+2b30KFXug=",
  "GDepth:Units": "mm",
  "GDepth:MeasureType": "OpticalAxis",
  "GDepth:ConfidenceMime": "image/png",
  "GDepth:Confidence": "acdbpQQ1lFTy/4Tp8Epx670E5QW5NwkNR+2b30KFXug=",
  "GDepth:Manufacturer": "Samsung",
  "GDepth:Model": "Samsung Galaxy S2",
  "GDepth:Software": "Truepic Foresight Firmware for QC QRD8250 v0.01",
  "GDepth:ImageWidth": 32.2,
  "GDepth:ImageHeight": 43.6
}
```


As defined by the GDepth specification, the following fields shall be present in all GDepth depth map assertions

- GDepth:Format
- GDepth:Near
- GDepth:Far
- GDepth:Mime
- GDepth:Data

17.13. Exif Information

The **Exif** assertion can be used to ensure that Exif information, for example about the capture device, is added to the asset in a way that can be verified cryptographically. It is preferable to copy this information from the `Exif` block or from the EXIF namespace of the XMP block. Information should be copied to JSON-LD using the XMP field names specified in the Exif 2.32 metadata for XMP specification. When copying the information from XMP, the data shall be re-serialized according to the rules of the JSON-LD serialization of XMP.

An Exif Information assertion shall have a label of `stds.exif`.

Any property from the latest version of the Exif specification (currently 2.32) can be added to the stds.exif assertion, with the exception of the MakerNote (37500, 0x927c) field, the contents of which are vendor-specific.

This assertion should be used to assert the precise location associated with the content using the Exif GPS properties (those starting `exif:GPS`). For broad information about the location, see Section 17.14, “IPTC Photo Metadata”.

An example of an Exif assertion:

```
{
  "@context" : {
    "dc": "http://purl.org/dc/elements/1.1/",
    "exifEX": "http://cipa.jp/exif/2.32/",
    "exif": "http://ns.adobe.com/exif/1.0/",
    "tiff": "http://ns.adobe.com/tiff/1.0/",
    "xmp": "http://ns.adobe.com/xap/1.0/",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  },
  "exif:GPSVersionID": "2.2.0.0",
  "exif:GPSLatitude": "39,21.102N",
  "exif:GPSLongitude": "74,26.5737W",
  "exif:GPSAltitudeRef": 0,
  "exif:GPSAltitude": "100963/29890",
  "exif:GPSTimeStamp": "2019-09-22T18:22:57Z",
  "exif:GPSSpeedRef": "K",
  "exif:GPSSpeed": "4009/161323",
  "exif:GPSImgDirectionRef": "T",
  "exif:GPSImgDirection": "296140/911",
  "exif:GPSDestBearingRef": "T",
  "exif:GPSDestBearing": "296140/911",
  "exif:GPSHPositioningError": "13244/2207",
  "exif:ExposureTime": "1/100",
  "exif:FNumber": 4.0,
  "exif:ColorSpace": 1,
  "exif:DigitalZoomRatio": 2.0,
  "tiff:Make": "NIKON CORPORATION",
  "tiff:Model": "NIKON D60",
  "exifEX:LensMake": "NIKON CORPORATION",
  "exifEX:LensModel": "17.0-35.0 mm",
  "exifEX:LensSpecification": { "@set": [ 1.55, 4.2, 1.6, 2.4 ] }
}
```


NOTE

The redaction process works in such a way that only an entire assertion can be redacted (see Section 5.7, “Redaction of Assertions”). If it is possible that future users may want to redact some information such as precise location, then it is recommended that multiple Exif assertions are generated. As described in Section 5.4, “Multiple Instances”, multiple instances of an assertion can be denoted by a double underscore and a monotonically-increasing number. For example, an assertion with label `stds.exif__1` could include the location data which may need to be redacted, and `stds.exif__2` can be used to include the camera information or other asset metadata.

17.14. IPTC Photo Metadata

The International Press Telecommunications Council defines a standard set of descriptive, administrative and rights metadata typically used by photographers, distributors, news organizations, archivists, and developers. Early versions of the standard used IPTC’s binary Information Interchange Model (IPTC-IIM) format and later versions introduced XMP support for all IIM fields, plus a new set of fields which can only be expressed in XMP. Together, these are called the IPTC Photo Metadata Standard.

The IPTC Photo Metadata assertion can be used to ensure that IPTC Photo Metadata, for example describing ownership, rights and descriptive metadata about an image, is added to the asset in a way that can be verified

cryptographically.

It is preferable to copy this information from the IPTC-IIM or XMP block. Information should be copied to JSON-LD using the XMP field names specified in the Exif 2.32 metadata for XMP specification. When copying the information from XMP, the data shall be re-serialized according to the rules of the JSON-LD serialization of XMP.

IPTC Photo Metadata properties can be copied to the C2PA assertion store as an assertion with the label `stds.iptc.photo-metadata`. The data should be in JSON-LD format using the XMP field names and structures specified in the IPTC Photo Metadata Specification.

Any property from the IPTC Photo Metadata Standard can be added to the `stds.iptc.photo-metadata` assertion.

In particular, the Location Created property can be used to assert the broad location associated with an asset. To assert the precise GPS location associated with an asset, the Exif information assertion should be used.

An example of an IPTC Photo Metadata assertion including location information:

```
{
  "@context" : {
    "Iptc4xmpCore": "http://iptc.org/std/Iptc4xmpCore/1.0/xmlns/",
    "Iptc4xmpExt": "http://iptc.org/std/Iptc4xmpExt/2008-02-29/",
    "dc" : "http://purl.org/dc/elements/1.1/",
    "photoshop" : "http://ns.adobe.com/photoshop/1.0/",
    "plus" : "http://ns.useplus.org/ldf/xmp/1.0/",
    "xmp" : "http://ns.adobe.com/xap/1.0/",
    "xmpRights" : "http://ns.adobe.com/xap/1.0/rights/"
  },
  "photoshop:DateCreated": "Aug 31, 2021", ①
  "dc:creator": [ "Julie Smith" ], ②
  "dc:rights": "Copyright (C) 2021 Example Photo Agency. All Rights Reserved." ③
  "photoshop:Credit": [ "Julie Smith/Example Photo Agency via Example Distributor" ], ④
  "plus:licensor": [ ⑤
    {
      "plus:LicensorName": "Example Photo Agency",
      "plus:LicensorURL": "http://exmplephotoagency.com/images/"
    }
  ],
  "xmpRights:WebStatement": "http://examplephotoagency.com/terms.html", ⑥
  "xmpRights:UsageTerms": [
    "Not for online publication. Germany OUT" ⑦
  ],
  "Iptc4xmpExt:LocationCreated": { ⑧
    "Iptc4xmpExt:City": "San Francisco"
  },
  "Iptc4xmpExt:PersonInImage": [ ⑨
    "Angela Merkel"
  ]
}
```

① Date Created

② Creator

③ Copyright Notice

④ Credit Line

⑤ Licensor

⑥ Web Statement of Rights

⑦ Rights Usage Terms

⑧ Location Created

⑨ Person Shown in the Image

NOTE	NOTE
	The redaction process works in such a way that only an entire assertion can be redacted (see Section 5.7, “Redaction of Assertions”). If it is possible that future users may want to redact some information such as precise location, then it is recommended that multiple `stds.iptc.photo-metadata` assertions are generated. As described in Section 5.4, “Multiple Instances”, multiple instances of an assertion can be denoted by a double underscore and a monotonically-increasing number. For example, an assertion with label `stds.iptc.photo-metadata__1` could include the location properties, creator name or other information which may need to be redacted, and `stds.iptc.photo-metadata__2` can be used to include other asset metadata.

17.15. Use of Schema.org

Schema.org is a collaborative, community activity with a mission to create, maintain, and promote schemas for structured data on the Internet. As such, they provide numerous JSON-based grammars that one may wish to include as an assertion. When using a schema.org type as a C2PA assertion, we require a full JSON-LD serialisation. As such, the top-level `@context` field must have a value of `http://schema.org`.

All Schema.org assertions shall have a label that starts with `schema.org` and be followed by the name of the schema that is being used. For example, `schema.org.ImageObject` can be used to include something extra about an image. Since schemas on Schema.org are not versioned as described here, no version indicator is used in their labels.

17.15.1. Claim Review

A schema that is used by the publishing community for web sites has recently also been introduced for images. A ClaimReview is used as 'A fact-checking review of claims made (or reported) in some creative work (referenced via itemReviewed).'

A ClaimReview assertion shall have a label of `stds.schema-org.ClaimReview`.

Inside the assertion (which, as described previously, is serialised as JSON-LD), the top-level `@type` field should be set

to a value of `ClaimReview`.

NOTE

The "Claim" in ClaimReview refers to a "claim made (or reported) in some creative work", not the C2PA claim block. ClaimReview assertions should be used as a way to mark the current asset as a review of the claims of another work, not any specific part of the C2PA manifest. To provide a review of a specific assertion, use an assertion metadata assertion.

An an example: an image infographic has been assembled that fact-checks some other claim that is currently in public distribution. The image infographic includes a ClaimReview assertion which references the text of the claim its reviewing, a fact-checking rating of the claim, and zero to many links to other works (e.g., articles, videos, images) that are making that claim.

The reviewRating property inside the ClaimReview assertion is of type Rating. Depending on how your organisation does fact-checking reviews, you may have a numerical rating, a true or false, or some other textual description of how the asset has reviewed a claim. https://www.claimreviewproject.com/user-guide provides a good guide on how to use the ClaimReview vocabulary.

A partial JSON Schema for this type is:

NOTE

This schema does not validate all aspects of the referenced schema.org types (just the top-level ClaimReview and Review properties). Please refer to https://schema.org for the details on referenced types, and https://json-ld.org/ for how to reference them in the assertion.

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "$id": "http://ns.c2pa.org/claim_review/v1",
  "type": "object",
  "description": "Assertion that describes a fact-checking review of claims made (or
reported) in some creative work (referenced via itemReviewed).",
  "examples": [
    {
      "@context": "http://schema.org",
      "@type": "ClaimReview",
      "claimReviewed": "The world is flat",
      "reviewRating": {
        "@type": "Rating",
        "ratingValue": "1",
        "bestRating": "5",
        "worstRating": "1",
        "ratingExplanation": "The world is not flat",
        "alternateName": "False"
      },
      "itemReviewed": {
        "@type": "CreativeWork",
        "author": {
          "@type": "Person",
          "name": "A N Other"
        },
        "headline": "Earth: Flat."
      }
    }
  ],
```

```
  "anyOf": [
    {
      "$ref": "https://json.schemastore.org/schema-org-thing.json"
    }
  ],
  "properties": {
    "claimReviewed": {
      "type": "string",
      "minLength": 1,
      "description": "A short summary of the specific claims reviewed in a ClaimReview."
    },
    "reviewAspect": {
      "type": "string",
      "minLength": 1,
      "description": "From http://schema.org/Review: This Review or Rating is relevant to
this part or facet of the itemReviewed."
    },
    "reviewBody": {
      "type": "string",
      "minLength": 1,
      "description": "From http://schema.org/Review: The actual body of the review."
    },
    "reviewRating": {
      "description": "From http://schema.org/Review: The rating given in this review. Note
that reviews can themselves be rated. The reviewRating applies to rating given by the
review. The aggregateRating property applies to the review itself, as a creative work.",
      "anyOf": [
        {
          "$ref": "https://json.schemastore.org/jsonld#"
        }
      ]
    },
    "itemReviewed": {
      "description": "From http://schema.org/Review: The item that is being
reviewed/rated.",
      "anyOf": [
        {
          "$ref": "https://json.schemastore.org/jsonld#"
        }
      ]
    }
  }
}
```

17.15.2. Creative Work

Schema.org provides a well-known and well-deployed set of types and metadata fields. One of the core types is CreativeWork, which is intended to describe any representation of creative effort. This assertion allows an asserter to provide various pieces of information about the asset, including who they are, and the date/time of publication.

A Creative Work assertion shall have a label of `stds.schema-org.CreativeWork`.

Inside the assertion (which, as described previously, is serialised as JSON-LD), the top-level `@type` field should be set to a value of `CreativeWork`. The JSON-LD document's root subject is the bound asset of the claim that this assertion is part of.

```
```

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "$id": "http://ns.c2pa.org/creative_work/v1",
  "type": "object",
  "description": "Assertion that describes the most generic kind of creative work, including
books, movies, photographs, software programs, etc.",
  "examples": [
    {
      "@context": [
        "http://schema.org/",
        {
          "credential": null
        }
      ],
      "@type": "CreativeWork",
      "datePublished": "2021-05-20T23:02:36+00:00",
      "publisher": {
        "name": "BBC News",
        "publishingPrinciples": "https://www.bbc.co.uk/news/help-41670342",
        "logo": "https://m.files.bbci.co.uk/modules/bbc-morph-news-waf-page-
meta/5.1.0/bbc_news_logo.png",
        "parentOrganization": {
          "name": "BBC",
          "legalName": "British Broadcasting Corporation"
        }
      },
      "url": "https://www.bbc.co.uk/news/av/world-europe-57194011",
      "identifier": "p09j7vzv",
      "producer": {
        "id": "https://en.wikipedia.org/wiki/Joe_Bloggs",
        "name": "Joe Bloggs",
        "credential": [
          {
            "url": "self#jumbf=c2pa/acme:urn:uuid:F9168C5E-CEB2-4faa-B6BF-
329BF39FA1E4/c2pa.credentials/Joe_Bloggs",
            "alg": "sha256",
            "hash": "Auxjtmax46cC2N3Y9aFmBO9Jfay8LEwJWzBUtZ0sUM8gA"
          }
        ]
      },
      "copyrightHolder": {
        "name": "BBC",
        "legalName": "British Broadcasting Corporation"
      },
      "copyrightYear": 2021,
      "copyrightNotice": "Copyright © 2021 BBC."
    }
  ],
  "anyOf": [
    {
      "$ref": "https://json.schemastore.org/schema-org-thing.json"
    }
  ],
  "properties": {
    "datePublished": {
      "type": "string",
      "minLength": 1,
      "format": "date-time",
      "description": "From http://schema.org/CreativeWork: Date of first
broadcast/publication."
    },
    "publisher": {
      "description": "From http://schema.org/CreativeWork: The publisher of the creative
```

```
work.",
      "$ref": "#/definitions/personOrgOrUri"
    },
    "identifier": {
      "type": "string",
      "minLength": 1,
      "description": "The identifier property represents any kind of identifier for any kind
of Thing (in this case, an identifier for this CreativeWork)."
    },
    "producer": {
      "description": "From http://schema.org/CreativeWork: The person or organization who
produced the work (e.g. music album, movie, tv/radio series etc.).",
      "$ref": "#/definitions/personOrgOrUri"
    },
    "creator": {
      "description": "From http://schema.org/CreativeWork: The creator/author of this
CreativeWork.",
      "$ref": "#/definitions/personOrgOrUri"
    },
    "editor": {
      "description": "From http://schema.org/CreativeWork: Specifies the Person who edited
the CreativeWork.",
      "anyOf": [
        {
          "$ref": "#/definitions/person"
        },
        {
          "$ref": "#/definitions/uri"
        }
      ]
    },
    "contributor": {
      "description": "From http://schema.org/CreativeWork: A secondary contributor to the
CreativeWork or Event.",
      "$ref": "#/definitions/personOrgOrUri"
    },
    "copyrightHolder": {
      "description": "From http://schema.org/CreativeWork: The party holding the legal
copyright to the CreativeWork.",
      "$ref": "#/definitions/personOrgOrUri"
    },
    "copyrightNotice": {
      "type": "string",
      "minLength": 1,
      "description": "From http://schema.org/CreativeWork: Text of a notice appropriate for
describing the copyright aspects of this Creative Work, ideally indicating the owner of the
copyright for the Work."
    },
    "copyrightYear": {
      "type": "integer",
      "description": "From http://schema.org/CreativeWork: The year during which the claimed
copyright for the CreativeWork was first asserted."
    }
  },
  "definitions": {
    "credential": {
      "type": "array",
      "description": "Verifiable Credentials (VC) associated with this actor.",
      "minItems": 1,
      "items": {
        "$ref": "http://ns.c2pa.org/hashed-uri/v1"
      }
    },
```

```
    "organization": {
      "type": "object",
      "description": "From http://schema.org/Organization: An organization such as a school,
NGO, corporation, club, etc.",
      "properties": {
        "name": {
          "type": "string",
          "minLength": 1,
          "description": "The name of the Organization."
        },
        "legalName": {
          "type": "string",
          "minLength": 1,
          "description": "From http://schema.org/Organization: The official name of the
organization, e.g. the registered company name."
        },
        "publishingPrinciples": {
          "type": "string",
          "minLength": 1,
          "format": "uri",
          "description": "From http://schema.org/Organization: The publishingPrinciples
property indicates (typically via URL) a document describing the editorial principles of an
Organization (or individual e.g. a Person writing a blog) that relate to their activities as
a publisher, e.g. ethics or diversity policies."
        },
        "logo": {
          "type": "string",
          "minLength": 1,
          "format": "uri",
          "description": "From http://schema.org/Organization: An associated logo."
        },
        "parentOrganization": {
          "description": "From http://schema.org/Organization: The larger organization that
this organization is a subOrganization of, if any.",
          "$ref": "#/definitions/organization"
        },
        "credential": {
          "$ref": "#/definitions/credential"
        }
      }
    },
    "person": {
      "type": "object",
      "description": "From http://schema.org/Organization: An organization such as a school,
NGO, corporation, club, etc.",
      "properties": {
        "name": {
          "type": "string",
          "minLength": 1,
          "description": "The name of the Person."
        },
        "givenName": {
          "type": "string",
          "minLength": 1,
          "description": "From http://schema.org/Person: Given name. In the U.S., the first
name of a Person."
        },
        "familyName": {
          "type": "string",
          "minLength": 1,
          "description": "Family name. In the U.S., the last name of a Person."
        },
        "credential": {
```

```
          "$ref": "#/definitions/credential"
        }
      }
    },
    "uri": {
      "type": "string",
      "minLength": 1,
      "format": "uri"
    },
    "personOrgOrUri": {
      "anyOf": [
        {
          "$ref": "#/definitions/organization"
        },
        {
          "$ref": "#/definitions/person"
        },
        {
          "$ref": "#/definitions/uri"
        }
      ]
    }
  }
}
```

As described in Section 7.3, “Using Credentials”, it is possible to associate a W3C Verifiable Credential for an actor with a specific role in the Creative Work via the `url` property of that role’s value. However, because the `url` value can be any valid URL, it is also possible to use any valid identity URL (e.g., OpenID or WebID).

Chapter 18. Open Topics

There are a number of open topics that have not been addressed in this public review draft specification. They are briefly described below, organized by the relevant section of the specification.

18.1. Assertions

- Support for marking a previous claim or assertion as "retracted", a feature commonly used in the news publication industry.
- Extending the prevention of assertion redaction to assertions other than C2PA-defined actions.

18.2. Binding to Content

- Add support for the binding of live video streams, very large HEIF and AVIF files, 3D formats, additional audio formats, raw text files and raw camera data.
- Addressing the case where servers make real-time modifications to content streamed to a client, potentially altering the cryptographic hash used to bind ISO BMFF content to a manifest.

18.3. Trust Model

- Supporting additional credential types for use in signing manifests.

18.4. Validation

- Consider whether a claim can be valid even if not all assertions are available, allowing applications and relying parties to adopt more stringent requirements when appropriate.
- Define a more structured approach to ingredient validation.
- Design for how a video player can communicate with the media validator to indicate that a discontinuity in video playback is expected, perhaps due to seek, fast forward, or other types of "trick play".

18.5. User Experience

- Working with the W3C and browser vendors to define a standardized model for exposing the provenance data to user agents.