Cowcerts specifications Documentation

Release 0.1

BTC Assessors

Jun 19, 2019

Contents

The **cowcerts standards** provide a specification for JSON-based data models to issue digitally verifiable credentials like educational certificates.

To do so, we leverage the work of:

- JSON as the open and widely-known data interchange format
- JSON Linked Data to define the meaning of JSON fields
- JSON Schema to ensure JSON documents fulfill the specification structure

. . . and other standards depending on the use case.

Note: We are now focused on **educational digital certificates** which contain the same information as the current analogic academic certificates, but providing all the advantages of digital technologies, including *digital signatures* and cutting edge *blockchain technologies* too.

CHAPTER 1

List of standards

Below you will find the specifications for the standards we're developing:

1.1 Cowcerts for Education

Cowcerts for Education provides a data model for digital certificates to prove that someone has completed their education successfully.

They aim to be a digital replacement for current educational certificates issued on paper, leveraging all digital technologies' advantages.

1.1.1 Features

Our goals with this standards are to provide:

- **Security**: Easy validation of the certificates using *digital signatures* and *blockchain technologies*. The identites of both the issuer and recipient are also easily and automatically validated.
- **Shareability**: The recipient of a certificate can easily share its educational achievements digitally using their preferred communication channels: social networks, email, instant messaging and much more.
- **Privacy**: The recipient is allowed to share just the information they want, as the certificate itself does not contain any personal information or any detail about the achievement like the grades obtained. Certificates' recipients can choose the amount information they share when sharing the certificate.

1.1.2 Base standards

Cowcerts for Education provides extensions to the following base standards:

- Open Badges standard
- Blockcerts standard

In order to get a basic comprehension of the standard, please read our abstract of the base standards:

Open Badges

Open Badges provides the JSON data model to represent achievements issued by some entity to a given recipient.

Data model

The core entities they define are:

Profile

Data model for an entity, either a recipient of an achievement, or its issuer. Provides details about them (name, description, image) and how to contact them (email, phone, URL).

Hint: For instance, a university that issues certificates like the *M.I.T.* or the recipient who completes a course.

Note: In practice, **Profile** is a data model just used for certificate issuers.

Note: Open Badges `Profile` Specification:

https://openbadgespec.org#Profile

Badge

Data model to represent an achievement, issued by an entity defined by the previous *Profile* data model. They provide information about the achievement obtained, the knowledge obtained, what was required to obtained the achievement and more.

Hint: For instance, a Computer Science degree by the M.I.T. could be a badge.

Note: It is also referred to as **BadgeClass**, but we call them **Badges** for short.

Note: Open Badges `Badge` (`BadgeClass`) Specification:

https://openbadgespec.org#BadgeClass

Assertion

Data model to issue a badge to a given recipient. Includes the badge issued, the recipient who receives the badge and data about the issuance, like data to verify if the assertion is valid or not.

Hint: For instance, a Computer Science degree by the M.I.T. issued to Silvio Micali could be an assertion.

Note: In previous versions it was called a **Badge** but this lead to overloaded meaning of the badge word. Please avoid using **Badge** to refer to a badge issued to a recipient and refer to it as an **Assertion**. The badge is just an achievement without any recipient.

Note: Open Badges `Assertion` Specification:

https://openbadgespec.org#Assertion

Endorsement

Data model to provide extra information about an *Assertion*, either to provide more information about the *Assertion*, comments about the existing information or validation information. The standard is open to receive any kind of extra information about a given assertion.

Note: Open Badges `Endorsement` Specification:

https://openbadgespec.org#Endorsement

Validation

In order to ensure that the certificates following this data model can be verified (to avoid fake certificates with impersonation), Open Badges provides two kinds of validations:

Hosted

The certificate is valid if is hosted on a trusted place.

Hint: For instance, a certificate issued by an issuer whose URL is https://mit.edu is valid just if the certificate can be downloaded from an URL related to the domain `mit.edu`

Note: Open Badges `HostedVerification` Specification:

https://openbadgespec.org#HostedBadge

Signed

The certificate contains an RSA signature using JWS. Therefore you can know that if the certificate signature is valid, the certificate was signed and therefore issued by the specified RSA public key.

Attention: The standard does not care about the real world™ identity of the holder of the private key. Therefore you can know whether the certificate was issued by a public key, but not who in the real world™ is behind the control of the signatures made by the private key matching that public key.

Using a PKI could be a solution for that, but that's not specified in the standard.

Blockcerts

The Blockcerts standard provides extension to the Open Badges standard to store the verification information of the certificate using *blockchain technologies* to enforce the integrity and allow a distributed timestamping of the issued certificates.

Data model

The data model is the same as the Open Badges one, adding the required data so that certificates validation can be stored in a *blockchain*.

Note: Details about the data model extensions are provided in the following document:

https://github.com/blockchain-certificates/cert-schema/blob/master/docs/schemas-2.1.md

---

SignatureLine

The only entity extended by Blockcerts which has no relation at all with storing and validating the certificates using *blockchain technologies* is the `SignatureLine`. A `SignatureLine` allows to place a handwritten signature picture in a certificate to improve its visualization.

Note: Blockcerts `SignatureLine` specification:

https://github.com/blockchain-certificates/cert-schema/blob/master/docs/signatureLineExtension_schema.md

---

Validation

In order to enforce the security of the certificate upon validating it, Blockcerts uses *blockchain technologies* as a way to store information to validate the certificate, including in it a hash and a digital signature.

Tip: Check this document for more information about the verification process:

https://github.com/blockchain-certificates/cert-verifier-js/blob/master/docs/verification-process.md

---

1.1.3 Specification

The specification comprehends the following documents:

JSON-LD Specifications

The following document provides a specification for the JSON-LD context names used in CowCerts JSON certificates.

All specifications here extend the Blockcerts context specifications (available in a markdown document)

And Blockcerts extends the Open Badges specifications

Entities

These are the extended JSON entities we define to achieve the digital granting of academic certificates. We also define the generic entities so that all the document can be understand within this single specifications document.

Assertion

Assertions are representations of an awarded badge, used to share information about a badge belonging to one earner. Assertions are packaged for transmission as JSON objects with a set of mandatory and optional properties. Fields marked in bold letters are mandatory.

Property	Expected type	Description
id	`IRI`	Uniquely identifies the assertion. Should be an `HTTPS` dereferenceable IRI so verifiers can link to the assertion. The document behind the IRI should provide information about the assertion and its validation.
type	JSON-LD type	JSON-LD assertion type. The type is fixed to: `Assertion`.
recipient	`IdentityObject`	The recipient of the achievement.
badge	`BadgeClass`	Badge document being awarded.
verification	`VerificationObject`	A collection of information allowing an inspector to verify this assertion.
issuedOn	`DateTime`	Timestamp of when the achievement was awarded. ISO 8601 compliant.
image	`ImageObject`	Image representing this achievement.
recipientProfile	`RecipientProfile`	Blockcerts extension allowing additional recipient details including recipient's public key to make a strong claim of the ownership over the key. Name must be set if the verifier provided by Blockcerts has to display it.
signature	`MerkleProof2019`	An extension that allows an issuer to issue an Open Badge on the blockchain and provide proof of inclusion in a blockchain transaction. This uses *Merkle Proof Signature Suite 2019*.
displayHtml	`Text`	HTML code to display the certificate Will be used in the blockcerts verifier to display the certificate visually. This way the certificate will always be visualized the same way even if the certificate displayer (in this case blockcerts verifier) changes.
issuedAt	`Text`	Place where the assertion was issued (the city for instance)
officialValidation	`Endorsement`	Mandatory field if the `BagdeClass` being awarded has an officiality requirement. This `Endorsement` has always a `MinistryClaim` it its claim field.
signatureLines	`Array of SignatureLine`	List of handwritten signatures that must appear in the certificate visualization
universalIdentifier	`Text`	Universal identifier of the assertion. Allows to identify the assertion uniquely in the universe. Contains a `UUID` string.

IdentityObject

A collection of information about the recipient of a `IdentityObject`.

Note: The Open Badges standard does not allow for a recipient to have more than one field (or a document) identifying it. For this reason, we use the government's tax ID in this identify object and all extra fields will be placed in an additional assertion, referencing this identity object.

To provide optional additional privacy to the recipient, this field will be always hashed.

Property	Expected type	Description
identity	`IdentityHash`	The hash of student email.
type	`Property` IRI	Fixed `email`, as Blockcerts specifies. Extra fields will be placed in an additional endorsement.
hashed	`Boolean`	Always `True`.
salt	`Text`	If the recipient is hashed, this should contain the string used to salt the hash. If this value is not provided, it should be assumed that the hash was not salted. This field is **mandatory**, as opposite to the Open Badges specification.

IdentityHash

A hash string preceded by a dollar sign ("$") and the algorithm used to generate the hash. The supported algorithms are MD5 and SHA-256, identified by the strings md5 and sha256 respectively.

Hint: For example:

`sha256$28d50415252ab6c689a54413da15b083034b66e5`

represents the result of calculating a SHA-256 hash on the string "mayze".

For more information, see how to hash & salt in various languages.

BadgeClass

A collection of information about a badge.

Property	Expected type	Description
id	`Text`	Unique IRI for the BadgeClass.
type	JSON-LD type	JSON-LD assertion type. The type is fixed to: `BadgeClass`.
name	`Text`	Name of the badge defined in the syllabus.
description	`Text`	Description of the knowledge acquired by a recipient of this badge.
image	`ImageObject`	Image representing the badge.
criteria	`@id:Criteria`	URI or embedded criteria document describing how to earn the achievement.
issuer	`Profile`	IRI or document describing the individual, entity, or organization that issued the badge.
signatureLines	Array of `SignatureLine`	List of visual signatures to display.
tags	Array of `Text`	An array containing tags that
legalText	`Text`	Additional information to the title that refers to its legality (legal text). describes the type of achievement.
official	`Boolean`	`True` if requires an official validation. Defaults to `False` if field is not present.

ImageObject

Metadata about images that represent `Assertions`, `BadgeClasses` or `Profile`.

These properties can typically be represented as just the id string of the image, but using a fleshed-out document allows for including captions and other applicable metadata. https://schema.org/ImageObject

Property	Expected type	Description
type	JSON-LD type	Type of the image object. Default is `schema:ImageObject`.
id	`IRI`	Data URI of the image.
caption	`Text`	Caption of the image, if any.

Criteria

Descriptive metadata about the achievements necessary to be recognized with an `Assertion` of a particular `BadgeClass`.

Property	Expected type	Description
type	JSON-LD type	Type of the image object. Fixed: `Criteria`.
narrative	`Text`	A narrative of what is needed to earn the badge.
id	`IRI`	The URI of a webpage that describes in a human-readable format the criteria for the `BadgeClass`.

Profile

A Profile is a collection of information that describes the entity or organization using Open Badges.

Property	Expected type	Description
id	`IRI`	Issuer blockchain address IRI, defined in issuerSchema by blockcerts.
type	JSON-LD type	Fixed: `Profile`.
name	`Text`	The name of the entity or organization.
url	`IRI`	The homepage or social media profile of the entity, accessible via HTTP.
telephone	`Text`	A phone number for the entity (E.164 format).
description	`Text`	A short description of the issuer, public or private and year of creation
image	`ImageObject`	An image representing the issuer.
email	`Text`	Contact address for the individual or organization.
revocationList	`URI`	HTTP URI of the Badge Revocation List used for marking revocation of signed badges. The revocation list is published as a JSON-LD document with type `RevocationList`.

RevocationList

List of assertions revoked by the issuer of the degree.

Property	Expected type	Description
id	`IRI`	The `id` of the RevocationList.
type	JSON-LD type	Fixed `Profile`.
issuer	`IRI`: `Profile`	The `id` of the Issuer.
revokedAssertions	`IRI`	A string `id` identification of a badge object that has been revoked. And a string `revocationReason` indicate the reason for revocation.

VerificationObject

Defined by verification property of https://w3id.org/openbadges#Assertion, with Blockcerts extensions for verification of badges on a blockchain.

Property	Expected type	Description.
type	`Array`	Fixed: `MerkleProofVerification2017`, `Extension`
publicKey	`Text`	Blockcerts extension: the expected blockchain address for the signer of the transaction containing the merkle proof. In Blockcerts publicKeys are typically represented with a `<scheme>:` prefix. For Bitcoin transactions, this would be the issuer public Bitcoin address prefixed with `ecdsa-koblitz-pubkey:`.

MerkleProof2019

Extends the Merkle Proof 2017 verification allowing to contain additional information about the assertion: a set of endorsements. They also contain extra information to verify the certificates in other blockchains other than the ones accepted by Blockcerts (for instance, the BlockValley ones).

Some endorsements are completely optional and are signed by their issuers independently so they cannot be included inside the assertion as when signing it the signature would contain the endorsements, which as we said are optional and may appear in the future.

Note: The *signature* field is the only field in an assertion that is not signed (because it contains the signature itself) therefore is the only place where these endorsements fit. They may be present or not to provide extra information about the assertion, but without them the assertion is valid.

This way we can package all information about the assertion, including its endorsements in a single portable document.

Property	Expected type	Description
type	`Array`	Composed type of `Extension` and `MerkleProofVerification2019`.
merkleRoot	`Text`	Batch identifier (SHA256).
targetHash	`Text`	Current document's SHA256 hash.
anchors	`Array` of *Anchor* objects	Array containing a list of anchor objects that define how the `merkleRoot` has been registered in one or more *blockchains*.
proof	`Text`	See Merkle Proof 2017 for more.
endorsements	Array of `Endorsement`	Endorsements containing additional information about the current document.

Anchor

Specifies how the `merkleRoot` in a `Merkle Proof 2019` signature has been registered on a *blockchain*.

Property	Expected type	Description
type	JSON-LD type	`BTCOpReturn` or `ETHData` defined in https://chainpoint.org/ .
sourceId	`Text`	Identifier, such as a transaction id, used to locate anchored data.
chain	`Text`	Chain is an optional field introduced by Blockcerts to help during verification. Check Merkle Proof Signature 2017 specifications document to see a list of accepted chains.
otherChains	`Array` of *Other chain* objects	Allows to verify the certificate in other chains other than the accepted by Blockcerts. This way Blockcerts can verify with a chain they consider valid and other verifiers may supply other *blockchains* to verify this on too.

Other chain

Allows to define another chain where an anchor may be placed other than the accepted by the Blockcerts standard to be used by validators that both understand Blockcerts and Cowcerts standards.

Property	Expected type	Description
id	`IRI`	Chain id. Should be able to be dereferrenced so the document behind provides more information about the chain and how to connect to it.
name	`Text`	Commonly used name to identify the chain.
protocol	`Text`	Protocol the *blockchain* uses. Valid values are `ETH` for *Ethereum*.
genesis	`Text`	Hexadecimal string representing the genesis block hash for the network.
consortium	`IRI`	Consortium where the chain belongs (if any).

Note: If the `id` is `urn:example:local`, the verifier of the certificate will connect against a local development *blockchain* so that *blockchain* checks can be triggered against a dummy *blockchain*.

In the case of protocol `ETH`, checks will be performed against a local *Ethereum* node using the `web3` JSON-RPC at default port 8545 against the same host where the verifier is running.

A note will alert the user that the *blockchain* checks are performed against a dummy blockchain and must not be taken seriously.

Endorsement

The endorsement class is very similar to an `Assertion`, except that there is no defined badge property. Instead, a claim property allows endorsers to make specific claims about other `Profiles`, `BadgeClasses`, or `Assertions`.

Property	Expected type	Description
id	`Text`	Unique IRI for the endorsement instance. If using hosted verification, this should be the URI where the assertion of endorsement is accessible.
type	JSON-LD type	Endorsement type; Fixed: `Endorsement`.
claim	`RecipientClaim` `EDSClaim` or `MinistryClaim`	An entity, identified by an id and additional properties that the endorser would like to claim about that entity. Three claim entities have been defined based on the attached information.
issuer	`Profile`	The profile of the endorsement's issuer.
issuedOn	`DateTime`	Timestamp of when the endorsement was published (`ISO8601` compliant).
verification	`Verification Object`	Instructions for third parties to verify this endorsement.
signature	`SignatureObject`	An extension that allows an issuer to issue an Open Badge on the blockchain and provide proof of inclusion in a blockchain transaction. This uses Merkle Proof 2017 Signature Suite.
signatureLines	`Array` of `SignatureLine`	List of handwritten signatures that must appear in the certificate visualization

Claims

MinistryClaim

Appends the information of the assertion's registry by the Education Ministry.

Property	Expected type	Description
type	`Array`	Composed type of `MinistryClaim` and `Extension`.
id	`IRI`	Id of the Assertion that endorsement is giving extra info.
ministrySignature	`SignatureLine`	Handwritten signature that gave validity to academic certificates analogically.
registryCode	`Text`	Alphanumeric code that represents the unique identifier of the official title record of the government of Andorra.

RecipientClaim

An extension of the information about the recipient of the assertion. Mainly uses fields from a Person

Property	Expected type	Description
type	`Array`	Composed type of `RecipientClaim` and `Extension`.
id	`IRI`	Id of the Assertion that endorsement is giving extra info.
givenName	`Text`	The name of the recipient
family-Name	`Text`	The surnames of the recipient
birthplace	`Text`	The place where the person was born.
birthdate	`Text`	The date when the person was born.
nationality	`Text`	Nationality of the person `ISO 3166-1- alpha2`.
nationalId	`Text`	The national ID person of their country, e.g. the CIF/NIF in Spain or NPI in Andorra.

EDSClaim

Adds additional information to the `Assertion` so it can be a valid European Diploma Supplement (EDS) .

Language codes must be compatible with `BCP47`. Think “en” or “es-MX”. JSON-LD allows much more expressive combinations of multiple languages in one document. It is likely that you may be able to produce Badge Objects taking advantage of these features that will not be understood by some or all validators or display tools. It is recommended to keep implementations as simple as possible and communicate with the standards group when you want to move beyond the example techniques expressed here.

It provides additional information to that included in the official degrees / diplomas and/or transcript, making it more easily understood, especially by employers or institutions outside the issuing country. (explicacio numero)

Property	Expected type	Description
type	`Array`	Composed type of `EDSClaim` and `Extension`.
id	`IRI`	Id of the Assertion that endorsement is giving extra info.
mainField	`Text`	2.2 Main field(s) of study for the qualification.
awardingInstitution	`Text`	2.3 Name (in original language) and status of awarding institution.
administeringInstitution	`Text`	2.4 Name (in original language) and status of institution administering studies.
language	`Text`	2.5 Language(s) of instruction / examination.
studiesLevel	`Text`	3.1 Level of qualification.
studiesLength	`Text`	3.2 Official length of programme.
access	`Text`	3.3 Access requirements.
mode	`Text`	4.1 Mode of study.
requirements	`Text`	4.2 Programme requirements.
grades	`Array` of `EDSSubject` items	4.3 Programme details.
gradingScheme	`Text`	4.4 Grading scheme.
qualification	`Text`	4.5 Overall classification of the qualification.
further	`Text`	5.1 Access to further study.
competences	Array of `Text`	5.2 Professional status and competences.
extraInfo	`Array` of `Text`	6 Additional information.
educationSystem	`ImageObject`	8 Information on the national higher education system Badge.
rectorSignature	`ImageObject`	7 Certification of the Supplement, image of the rector signature
managerSignature	`ImageObject`	7 Certification of the Supplement, image of the manager signature

EDSSubject

This object includes all the information related to one subject that will be included in the European Diploma Supplement (EDS).

Property	Expected type	Description
name	`Text`	Name of subject
choice	`Text`	Type of the subject, it could be one of the following: `OB`: Obligatory; `OP`: Optional; `LL`: Free Choice
semester	`Text`	Semester during which the studies were taken
year	`Number`	Year when the subject was taken.
mobility	`Text`	International academic mobility `M` if was studied abroad, – if not.
grade	`Text`	Grade (`CO`: if convalidated) Otherwise, the grade obtained as a float. (ie: `8.4`). Using a dot `.` as decimal separator.
credits	`Number`	Amount of ECTS credits the subject took.

JSON Schemas

We provide a series of JSON Schema documents to allow implementors of the standard ensure their documents follow the structure specified by the standard.

https://gitlab.com/cowcerts/schemas/tree/master/cowcerts/

> In that folder, choose a version folder, and you'll find the schemas inside > the `jsonschema` folder

You can use our *Python* script to check a document against our schema.

Just run

```
python -m validator.py
```

For more information.

Validation

To validate the educational digital certificates' JSON documents, the process described in the following document must be performed.

Tools

This process is automatically performed by this JavaScript library:

https://gitlab.com/cowcerts/libverifier-js

And a visual verification can be triggered using a WebComponent:

https://gitlab.com/cowcerts/verifier

Process

Blockcerts Validation

The first validation step is to perform all validation steps as specified by the Blockcerts validation process.

This validation has to be performed for all documents included in the certificate, including the endorsements included in the signature field.

Cowcerts Validation

TODO

1.2 Contributions

Please, feel free to contact us with your ideas and feedback to the following email address:

1.3 Indices and tables

- genindex
- modindex
- search