eHealth Network

Guidelines on

Technical Specifications

for EU Digital COVID Certificates

Validation Rules

V1.1

2022-02-23

The eHealth Network is a voluntary network, set up under article 14 of Directive 2011/24/EU. It provides a platform of Member States' competent authorities dealing with eHealth.

Adopted by the eHealth Network on 23.02.2021.

-Keep this page free-

Contents

Version History

Version	Date	Description
1.0	2021-06-09	First version
1.1	2022-02-23	Minor corrections

This document complements normative technical specifications adopted and published as Commission Implementing Decision (EU) 2021/1073 (with any amendments, such as Commission Implementing Decision (EU) 2021/2014). The document should be read together with the legal acts.

1. Terminology

Definition	Description
Technical check	Technical check (rules) on the authenticity, integrity, structure and time stamps of the QR code. These are not specified in this document.
Business Validation	Business rule validation checks on the DCC payload against acceptance and invalidation rules.
Verification Datetime	Verifcation Datetime is the date+time against which the rules are checked. For instance: date of departure, date of arrival, current date etc.
CoD	Country of Departure
CoA	Country of Arrival
DCC	EU Digital COVID Certificate
FFT	Fit for Travel, acceptance for access to a MS <<term may change>>
MS	Member State or Member States
Holder	A Holder of a DCC.
Proof	A cryptographically signed digital assertion of a vaccination, test result or recovery status of a holder.
Verifier	A verifier uses trusted cryptographic information of an issuer to verify the proof of a holder.
Issuer	The issuer issues or signs proofs about a holder statement.
Rule Engine	A rule engine processes rules over a set of data defined in a standardised manner.

2. Overview

The EU DCC Validation Rules are applied on the payload of the DCC. All "technical" validations have to be performed in the verifier applications to ensure that these checks are not overridden. This includes:

- Check of EXP Date
- Check of the Data Format (CBOR, Schema)
- Check of the Cryptographic Signature

All checks that are based on the semantics of the DCC payload must be performed as validation rules to ensure the exchange/interoperability of this information to other countries (hard coded rules on the payload cannot be explored by others). This should ensure the following behaviour:

- All wallet apps of all countries must be able to evaluate the current rule set of a country
- All verifier apps of all countries must be able to evaluate the current rule set of a country
- External parties (e.g. Reopen Europe, Airlines, Websites) must be able to use the rules as an input for checklists, visualisations etc.
- All verifier apps must be able to select a rule set to be used for the check against a scanned DCC
- All wallet apps must be able to select a rule set to be used for the check against a selected DCC

Under this site conditions, it's not recommended to implement hard coded rules on the payload, otherwise these rules are "hidden" to others.

3. User Stories

3.1 Departure

A user with the verifier app in the CoD wants to cross check, if a holder of DCC fulfils all requirements of the CoA. The user opens the verifier app, selects the CoA (e.g. from the boarding pass) and scans the provided DCC. The verifier app performs the technical validation. If this validation was successful, the verifier app performs the logical validation by checking the payload for additional rules set by the CoA. If all are passed, the verifier app shows green and valid.

3.2 Arrival

A user with the verifier app in the CoA wants to cross check, whether a holder of DCC fulfils all requirements of the CoA. The user opens the verifier app. The CoA is his default setting for scanning the provided DCCs. The verifier app performs the technical validation. If this validation was successful, the verifier app performs the logical validation by checking the payload for additional rules set by the CoA. If all are passed, the verifier app shows green and valid.

3.3 Booking

The booking website needs to check before a successful booking the validity of a DCC. For this purpose the service needs access to the logical rules to ensure the validity of the provided data.

3.4 Holder Information

A holder of a DCCs needs to check in the wallet app, whether their current certificate is still valid for entering another MS. For this purpose, the user selects a DCC and selects a MS to which the DCC should be checked against. The result is a list where all rules are provided with green arrows or red crosses (included the most important technical validations), so that the user knows exactly what kind of actions he has to perform. For instance: if the test certificate is not more valid, the checklist shows that the date time is expired, but the type of the test is still ok. A holder must have all the time a full overview of his current state.

3.5 Public Information

Some service providers want to visualise checklists or information about the current situation in a MS. These service providers are using the defined rules to create a human readable presentation of the current situation. E.g. a visitor clicks on a state and gets the actual rule set as a checklist.

4. Principles

4.1 Acceptance Rules

Acceptance Rules are all rules to generate acceptance for the DCC. To standardize the interpretation of a rule and the interoperability meaning, each defined rule must respect the following principles:

1. The rules apply only for the trip from departure to arrival. After arrival, extra rules may apply in the CoA, such as quarantine, extra tests et cetera. These are not in scope of the FFT check.
2. Besides QR code data elements, the following external information elements are to be used in the FFT check:
    1. Verification DateTime (date and time against which the data in the QR code are compared.). If no Verification DateTime is provided, it will be filled with the current date and time.
    2. Country of Destination (ISO 3166, 2-character)
       Other external parameters such as the age of the citizen, "color" of Country of Departure are **not in scope.**
3. The QR code contains information on **1** event with one entry: either a vaccination, a negative test, or a recovery statement (V, T or R).
4. The rules are checked against the Verification DateTime.
5. Rules must have a valid starting date and end date, indicating between witch time points they are active. The end date may be open.
6. Only the active rules for the CoA are checked, and only the rules for the specific type of certificate (V, T or R).
7. New rules come into effect at least 48 hours after they have been uploaded to the DCC Gateway.
8. MS create their own set of rules. The EU rules are just for information and must be included or edited in the MS rule set. This avoids unclarity or conflicting rules between EU and MS rules: only MS rules will be processed.
9. All rules for a country are processed and all must resolve to true.
10. JSON Logic Syntax of a specified version is used to define the rules.

4.2 Invalidation Rules

Invalidation Rules are all rules defined by an issuing country to invalidate DCC. These rules cannot be overridden by a verifier country and must be executed. To standardize the interpretation of a rule and the interoperability meaning, each defined rule must respect the following principles:

1. All rules from the issuing country must be processed.
2. The rules will be applied against the entire DCC payload.
3. JSON Logic Syntax of a specified version is used to define the rules.
4. GDPR Rules must be respected.

4.3 Dates and Time Handling

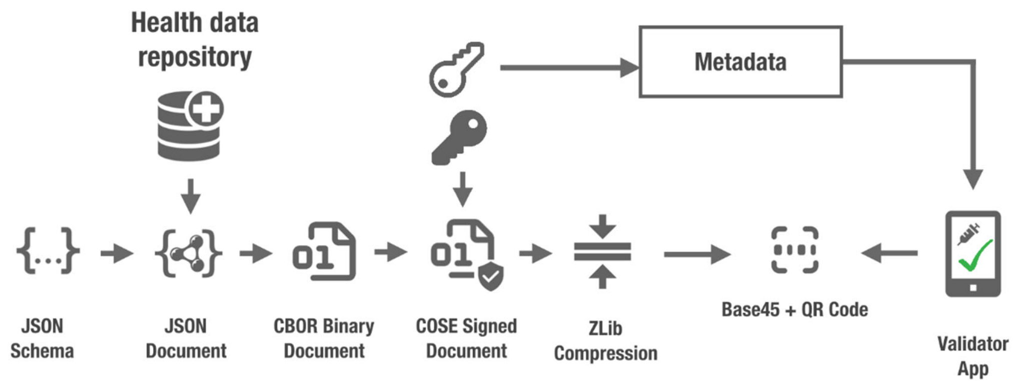
For a proper handling of any kind of rules an appropriate date and time handling is strictly necessary. For this purpose the principles for handling dates and time are:

1. The signing certificate expiration datetime supersedes the expiration datetime in the DCC.
2. Expiration datetime of the DCC supersedes the end date of the respective V/T/R Dates.
3. All used times inside the DCC must be time zoned for a proper comparison.

5. Rule Processing

5.1 Overview

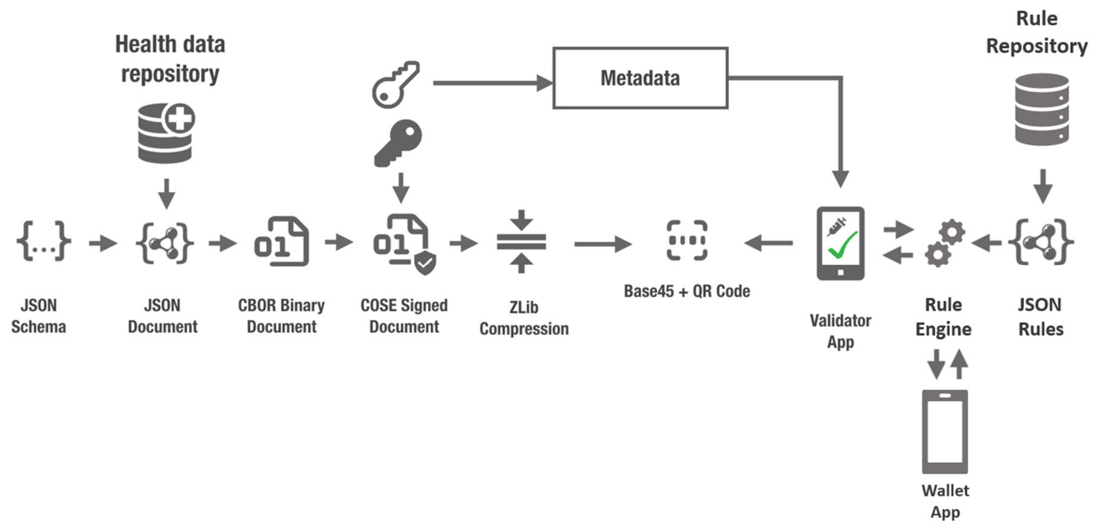
The basis for the rule processing is the default validation defined in the DCC Specification, where a validator app scans a QR code and extracts the DCC Payload.



The rule processing is built on top of the standard data validation flow defined by the DCC Specification. To ensure the maximum interoperability for applied rules between countries, the processed payloads must meet the following criteria:

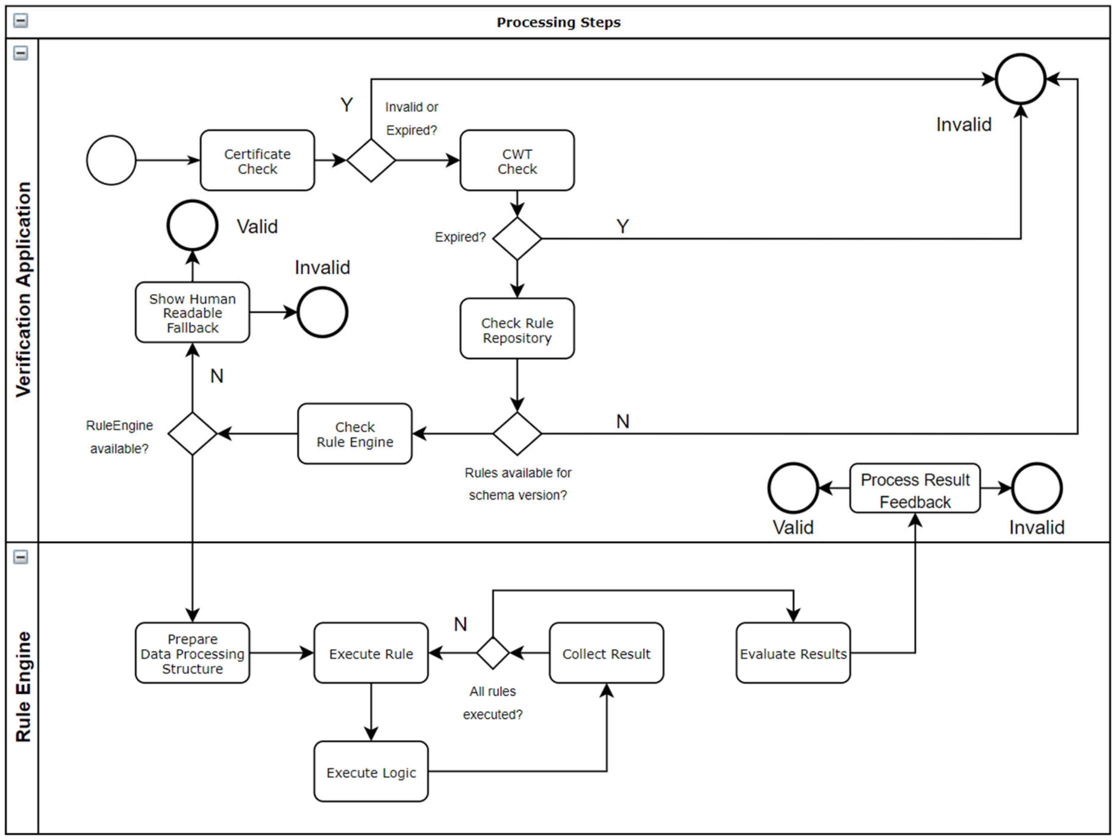
1. Each QR Code contains only one type of Proof (v/t/r)
2. The Proof contains only one event with one entry (v/t/r)
3. The Rule Engine processes only the QR Code Content and some external standardised parameters

This guarantees a proper processing of rules on a single object without side effects or mixing of edge cases. The validator and the wallet app can then be connected to a rule repository to provide this information to fulfil the user stories described above.



5.2 Processing Steps

The rule processing contains some common steps before the rules can be applied. At first there must be a technical validation of the received certificate for expiration and schema compatibility. If this is given, the process can continue with the rule engine check-up. If the engine is supported (backwards compatibility must be given), then the processing can be started. Otherwise human-readable feedback must be created. The second step is a logical validation by the rules engine, which processes the rules with prepared data for a certain certificate type. This prepared data must contain external data from the technical environment, for instance the current time, the current country code for validation and other standard values to give the rule engine a context for the verification.



The output of the rules must be collected to get a full picture of the final result. This is necessary for a proper feedback to the verification app which can handle then the complete result matrix. The result matrix should contain at least the Rule Identifier, the current values and the Boolean result value. In the case of a validation error, a verification application (including the wallet) should feedback the validation result to the user similar to this table:

Rule Identifier	Rule Description	Result	Current
VR-XX-00123	Number of doses must be 2/2	false	Dose Number: 1 Total Series of Doses: 2
VR-XX-00111	Valid Vaccine	True	vaccine medicinal product: Astra Zeneca

5.3 Selective Processing

The rule processing must be selectable in the wallet and in the verifier app to decide which rules of which country has to be applied against a payload. For this purpose both apps must have a selection by country to give the user the possibility to check-up against different rule sets.

5.4 Fallback Scenarios

5.4.1 Incompatible Rule Engine Versions

In the case of an incompatible engine version (rule engine version> current version), the technical validation has to be performed a human readable fallback. This fallback can be a table similar to this table:

Rule identifier	Rule description	Result	Current
VR-XX-00123	Number of doses must be 2/2	OPEN	Dose Number: 1 Total Series of Doses: 2
VR-XX-00111	Valid Vaccine	OPEN	vaccine medicinal product: Astra Zeneca

Each of the results has to be decided manually by the operator. For instance, by selecting each row and giving a check mark or by simply confirming that the check has successful passed. This check-up is a completely manual check-up, because the engine is not able to perform automatically a check with a rule.

This table can be presented similar to that table:

Rule identifier	Rule description	Result	Current
VR-XX-00123	Number of doses must be 2/2	OPEN	Dose Number: 1 Total Series of Doses: 2
VR-XX-00111	Valid Vaccine	True	vaccine medicinal product: Astra Zeneca

The wallet app should show the same table, but with a warning for each open rule, because this can lead to time delays during the travel.

5.4.2 Incompatible Schema

In the case of a schema version provided by the DCC that is unknown or not supported by the verification application, the verifier application can only present the complete content and let the verifier decide.

5.5 Rule Format and Repository

5.5.1 Rule Identifier Pattern

5.5.5.1 Acceptance

Rule Prefix	-	Country Code	-	Unique Sequence Number
GR	-	EU	-	0001
"GR" for General Rule **"VR" for Vaccination Rule** **"TR" for Test Rule** **"RR" for Recovery Rule**		ISO 3166 OR "EU"		Incrementing Number

5.5.5.2 Invalidation

The invalidation identifiers can be chosen, e.g. a GUID.

5.5.2 Rule Format

All rules are technically defined in a JSON format which describes the type, validity, the used versions, the human readable descriptions etc. The table below describes which fields must be present in the JSON to describing a rule.

Field	Description	Datatype	Example values
Identifier	The unique rule name	string	GR-CZ-0001
Type	Type of the Rule.	string	Acceptance/Invalidation

Version	Version of the Rule	string	1.0.0 (semver)
SchemaVersion	Version of the used Schema	string	1.0.0 (semver)
Engine	Type of the RuleEngine	string	"CERTLOGIC"
EngineVersion	Version of the used Engine	string	2.0.1 (semver)
CertificateType	Type of the certificate	string	General, Test, Vaccination, Recovery
Description	Array of Human readable description of the rule	Array	[{“lang”:“en”,desc:"Full vaccination course 2/2"}]
ValidFrom	Start Validity of the Rule	string	DateTime value
ValidTo	End Validity of the Rule	string	DateTime value
AffectedFields	Fields of the payload which are used by the rule.	Array	["tg", "mn"...]
Logic	The logic payload in JSON.	Object	JSON

An Example of the structure can be found in Appendix A

5.5.3 Repository

All rules are stored in a repository which must observe the validity time of a rule. The rules for Invalidation and Acceptance are stored in separated sections of the repository. If the "ValidTo" is expired, the rule is automatically deleted, if the owner of the rule does not update the rule.

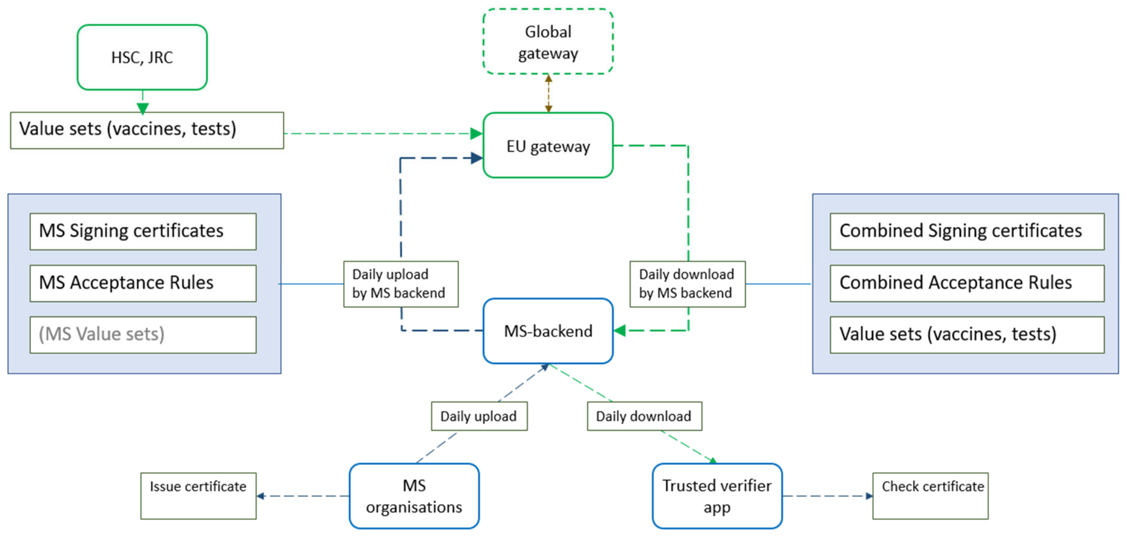
The repository performs content checks on the uploading/updating of rules to avoid conflicts:

Check	Description	Exception if not valid
UploaderCorrect	Check if the uploading Country uses the right format for the identifier. Country is extracted from Upload certificate.	Rule cannot be set for another country.
RuleValidFrom	The validFrom date must be minimum 48 hours in the future on upload time. If the rule already exists, the valid from is taken from the old entry.	Rule cannot be enabled within a time window less than 48 hours.
RuleAlreadyExpired	The validTo must be minimum five in the future in relation to the validFrom date.	Rules with validity less than 72 hours are not accepted.
DescriptionFilled	Human Readable Description must be filled with minimum one entry with 20 signs and language “en”.	Description must be filled for fallback.
VersionChecks	The inserted Versions must be semVer	Invalid Version Number
SchemaCheck	The schema of the rule must be valid.	Invalid Schema.
LanguageCheck	The language in the description array must be a valid one.	Invalid Language.

6. Rule and Valueset Distribution

6.1 Functional Overview

The rules and value sets are provided over the EU gateway additionally to the existing functionality of up/downloading signer certificates. The rules can be uploaded by each member state, but the value sets are maintained centrally by the eHN. In a future version the value sets can be uploaded by the member state as well, but this needs further alignment and harmonization of the value set interpretation.



6.2 Rules API

Route	Action	Description	Parameters	Response content
/rules /{country}	GET	Returns a List of Rules for a Country.	Country	Array of CMS Messages
/rules /{ruleIdentifier}	PUT	Upload/Update of a single Rule	CMS Message/RuleIdentifier	-
/rules /{ruleIdentifier}	DELETE	Deletes a rule.	RuleIdentifier/Signature	-
/countrylist	GET	List of onboarded Countries	-	Array of Country Codes

Note: The rules are immutable. After uploading it can be only deleted if a new version is uploaded before.

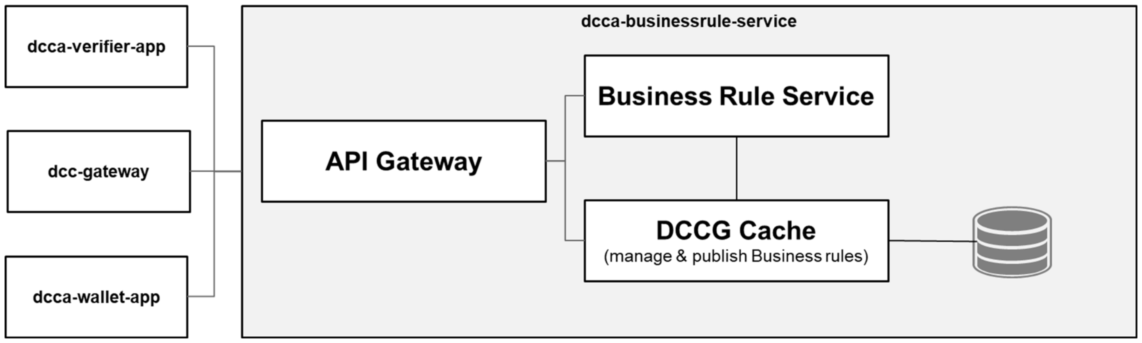
6.3 Valueset API

Route	Action	Description	Parameters	Response content
/valuesets	GET	ID List of available Valuesets	-	JSON Array
/valueset/{id}	GET	Valueset for a certain ID	ID as Path Parameter	JSON

The valuesets will be placed as JSON on the server and will be published by a public route.

6.4 App Provisioning

The verifier and the wallet app get the business rules from a business rule backend which downloads the rules from the gateway.



7. Rules Engine

7.1 Overview

The rule engine has the task to prepare the extracted JSON payload for a standardised rule processing. This contains a standardised container format, harmonised timestamps and the insertion of external parameters to have a standardised set of parameters that are identical across all rules and a standardised rule syntax.

7.2 Rule Syntax

The rule syntax is defined by JSON Logic[1] which defines a set of operators in JSON format. For this syntax are already implementations available in the most common languages. Some of that languages differ in the interpretation of this syntax; therefore the behaviour must be always aligned. For this purpose, the CertLogic subset of the JsonLogic Syntax is defined to align the behaviour across all implementations (see Appendix C).

7.3 Dates and Time Handling

The engine must convert all timestamps and dates to ISO8601 time zoned dates before processing to ensure a proper comparing and calculation of date values. For this purpose, the implementation must provide a custom operator "plusDays" to add and substract days of a given date as in this proposal:

```
jsonLogic.add_operation("plusDays", function(date,days){
let d = new Date(Date.parse(date));
d.setDate(d.getDate() + days);
return d.toISOString();
});
```

It has also been ensured that the implementation is able to realize the "between" operator for dates correctly which is defined in the JSON Logic Javascript Version as a compare operation between other values.

7.4 Basic Data Processing Structure

The basic data structure must follow a standardised manner to ensure that each rule defined in a special version is processed correctly. This structure contains a block for external data and the DCC content extracted by the rule engine. All rules are applied on this structure to have a standardised syntax for each rule. The format is defined in JSON and the order of the fields can be arbitrary.

Field name	Description	Format
external	Block for external parameters	JSON
payload	JSON Payload of the DCC	JSON

The payload contains the extracted entry of the certain certificate type (e.g. one entry of a vaccination). If the type is "general" the payload contains the complete DCC.

7.5 External Parameters

All external parameters are inserted by the rule engine which extract these parameters from the CWT, the app environment or from other available variables.

[1] https://jsonlogic.com/

Parameter	Description	Validation fieldname	Type	Value
Validation Date	Evaluation Timestamp which is used to check the validity (Reference Date)	validationClock	Date Time (ISO 8106)	"2021-04-21T18:25:43-05:00"
Value Sets	Lists of values defined by each valueset id.	valueSets	JSON Object with string arrays of each valueset.	"disease-agent-targeted":["840539006"]
Country Code	Country Code of country of interest (can be CoA or CoD)	countryCode	String (ISO 3166)	"NL", "CZ",...
Expiration Date	The expiration date of the certificate "exp" Field.	exp	String (ISO 3166)	"2022-04-21T18:25:43-05:00"
Issuing Date	The issuing date of the certificate "iat" field.	iat	String (ISO 3166)	"2022-04-21T18:25:43-05:00"

An example can be found at the end of this page in Appendix B.

JSON Logic Usage:

{ "in":[{"var":"hcert.v.0.tg"},{"var":"external.valueSets.disease-agent-targeted"}] }

7.6 Validation Logic

The validation logic must execute ALL rules for a specific country. All rules must be processed to provide a complete overview about the state of a DCC. The final result MUST be true or false (True=VALID, False=INVALID). A mixed state should not be allowed to express the rules as strict as possible. The logical pattern is:

Result = AND[Rule1, Rule2,Rule3]

All rules must be created in a way that the result expresses one validation for one entity. For instance, one rule for Vaccination Status, one rule for Validity Status. Conflicting Rules like "Dose1/2"+"Dose2/2" should be avoided, because they cannot be applied on one certificate in the same time (E.g. "CheckDoses" is one entity and can perform multiple checks inside).

8. Predefined Rules

8.1 Available Data Fields

This list of data fields is currently available in the DCC Schema (version 1.0.1).

Data field	Abbreviation	Type
Target Disease Agent	tg	Vac/Test/Rec
Vaccine or Prophylaxis	vg	Vac
Vaccine medicinal product	mp	Vac
Marketing Authorization Holder	ma	Vac
Dose Number	dn	Vac
Total Series of Doses	sd	Vac
Date of Vaccination	dt	Vac
Country of Vaccination	co	Vac
Certificate Issuer	is	Vac/Test/Rec
Unique Certificate Identifier: UVCI	ci	Vac/Test/Rec
Type of Test	tt	Test
NAA Test Name	nm	Test
RAT Test name and manufacturer	ma	Test
Date/Time of Sample Collection	sc	Test
Date/Time of Test Result	dr	Test
Test Result	tr	Test
Testing Centre	tc	Test
Country of Test	co	Test
ISO 8601 Date of First Positive Test Result	fr	Rec
ISO 8601 Date: Certificate Valid From	df	Rec
Certificate Valid Until	du	Rec

Appendix A - examples

```
{
"Identifier": "GR-CZ-0001",
“Version”:"1.0.0",
“SchemaVersion”:"1.0.0",
"Engine":"CERTLOGIC",
"EngineVersion":"1.0.0",
"Type":"Test",
"Description":"The Field “Doses” MUST contain number 2 OR 2/2.",
"ValidFrom":"2021-05-27T07:46:40Z",
"ValidTo":"2021-06-01T07:46:40Z",
"AffectedFields":["dt","nm"]
"Logic":{
            "and": [
{“>=":[ {"var":“dt", "23.12.2012" ]}},
{“>=":[ {"var":“nm", "ABC" ]}}]
}
```

Appendix B

JSON Example before JSON-LOGIC Processing:

```
{
  "external":{
                    "validationClock":"2021-10-21T18:25:43-05:00",
                    "valueSets"  : {
                              "test-type":[...],
                              "test-result":["260415000","..."]
                              },
                    "countryCode":"CZ",
                    "exp":"2022-10-21T18:25:43-05:00"
          },
"hCert":{
          "v":[ {
                    "tg": "840539006",
                    "tt": "LP6464-4",
                    "ma": "1331",
                    "sc": "2021-05-15T12:34:56Z",
                    "dr": "2021-05-16T12:45:01Z",
                    "tr": "260415000",
                    "tc": "Testing center 1",
                    "co": "RO",
                    "is": "Ministry of Health",
                    "ci": "URN:UVCI:01:RO:QW3L2LL66Q#4"
          } ]
          }
}
```

Appendix C - CertLogic

CertLogic specifies a *subset* of JsonLogic Syntax, where necessary extended with custom operations - e.g. for correct handling of dates.

Regarding the relation of CertLogic with JsonLogic, and the DCC validation rules:

- Any DCC validation rule must be expressed according to CertLogic, and **may not** use JsonLogic in a way that's inconsistent with CertLogic, or not provided for by CertLogic. In particular, only JsonLogic constructs present in this CertLogic specification may be used.
- The use of "truthy"/"falsy" values that are not already JSON Booleans is discouraged, and limited. In particular, DCC-validation rules should evaluate to a JSON Boolean, not to any "truthy"/"falsy".
- The semantics of the validation rules may be *expressed* in CertLogic, but are **not** (strictly) *defined* by it: all implementations should pass the unit tests assertions defined for the validation rules, and should pass additional tests that ascertain the intent of the validation rule.
- This specification can be expanded when the needs arises, but never shrunken.
- The intention is that CertLogic remains compatible with JsonLogic, but may expand on JsonLogic when and where necessary.
- In the interest of testability and risk mitigation, CertLogic is kept as small and simple as possible, without any "programmers' convenience".

The semantics of CertLogic is that of a function which takes a *CertLogic expression*, a *data context*, and returns a value. All three of these values are JSON values that must be *self-contained*, and *non-referential*. This means that:

- These JSON values can be finitely serialised, and identically deserialised.
- JSON objects anywhere in any of these values have at most one incoming reference (that of their parent), and no outgoing references, except to children.

CertLogic *logical* expressions are of the following form, *except* for data access (var) - see below.

```
{
  "<operation id>": [
    <first argument>,
    <second argument>,
    // ...
    <last argument>
  ]
}
```

The CertLogic function always evaluates to some value, and never throws an error. Operating on a value null evaluates to null in most, but not all, cases.

A JSON array evaluates to an array with every item evaluated separately. If a JSON object (not an array) is not of the logical expression form above, and also not of the var-form, it evaluates to itself.

Truthy and falsy

Allowed falsy values are: false, and null. Allowed truthy values are: true, any non-empty array, and any object. Using other values that are "traditionally" falsy or truthy is regarded as undefined behaviour.

The reason to do this is that JsonLogic has a notion of truthy/falsy that differs from the usual JavaScript one, precisely to aid in cross-platform adoption. CertLogic restricts this notion even further to avoid confusion, or unjust reliance on behaviour that's perceived as “common” but isn't (cross-platform).

Literals: arrays, booleans, integers, and strings

The usual array, boolean, integer (as a subset of JavaScript's Number type), and string literals are usable. Literal for the following (types of) values are not allowed: objects, null, and dates.

A datetime (or timestamp) has to be constructed by performing a plusDays operation on a string with 0 days added. This makes it possible to ensure consistent datetime representations across platforms, without being able to implicitly rely on the behaviour of native datetime types in combination with the other (allowed) operations.

Data access (var)

The data context can be accessed through an operation of the following form:

```
{ "var": "<path>" }
```

The <path> is a string containing a path that “drills down into” the current data context. It must be composed of “path fragments” which are either “Lispy” words (starting with a word character followed by any number of word or number characters, or hyphens), or integers, and which are separated by periods (.). It must match the regular expression /^((\w[\w\d-]*)|\d+)(\.((\w[\w\d-]*)|\d+))*$/.

In terms of most mainstream programming languages: if it is a variable/slot holding the current data context, then { "var": "<path>" } essentially means it.<path>. Data access of a null or non-existing value evaluates to null.

An integer path fragment signifies array indexing. Array indices are 0-based, and accessing a non-existing index evaluates to null.

The empty path "" serves as a shorthand for accessing the entire data context - alternatively, you can read it as it (or this). The variant { "var": <integer index> } is superfluous as { "var": "<integer index>" } achieves the same result. The variant which provides a default value instead of a missing/null result is (currently) not allowed.

As noted before: the var data access operation is the only type of expression that can have a non-array argument specification.

If-then-else (if)

Conditional logic can be implemented through an operation of the following form:

```
{
  "if": [
    <guard>,
    <then>,
    <else>
  ]
}
```

If the <guard> evaluates to a truthy value, then the <guard> expression is evaluated, otherwise the <else> expression.

Operations with binary operators

The following binary operators from JavaScript are available: ===, and, >, <, >=, <=, in, +.

An operation with a binary operator has the following form:

```
{
  "<operator>": [
    <operand 1>,
    <operand 2>,
    ...,
    <operand n>
  ]
}
```

For the ===, in, and + operators, n must equal 2.

The in operator checks whether <operand 1> is a member of <operand 2>, which must be an array - possibly empty. (This must be checked beforehand through other means.)

The *and* operator can be used *variadically*: it can have any number of operands greater than 1. An operation with the *and* operator returns its first operand that evaluates to a falsy value, or the evaluation of the last value.

The comparison operators >, <, >=, <= (exempting equality ===) can be used in (the customary) binary form, or in the ternary form, with n equal to 3. The ternary form

```
{
  "<op>": [
    <operand 1>,
    <operand 2>,
    <operand 3>
  ]
}
```

has the following semantics: (<operand 1> <op> <operand 2>) and (<operand 2> <op> <operand 3>). The operands must be comparable values: integers, strings, or datetimes.

Negation (!)

The negation operation takes one operand, and returns true if that operand's falsy, and false if it's truthy.

Offset datetime (plusDays)

A datetime offset operation has the following form:

```
{
  "plusDays": [
    <operand that evaluates to a string with a datetime in the allowed format>,
    <integer: the number of days to add (may be negative)>
  ]
}
```

This operation is the *only* way to construct datetime values.

Reduction (reduce)

A reduction operation has the following form:

```
{
  "reduce": [
    <operand>,
    <lambda>,
    <initial>
  ]
}
```

The <operand> must be an array - possibly non-empty. This must be checked beforehand by other means. Often, the expression { "var": "<operand>.0" } can be used to check that <operand> is a non-empty array.

The reduce operation is equivalent to a left-fold over the array <operand> with <initial> prepended, using the provided <lambda> function. That function is provided with a modified data context of the form { "current": <current>, "accumulator": <accumulator> }, with the <current> equalling the items of the array (in that order), and <accumulator> equalling the result of the left-fold so far. In particular, the reduce of an empty-array <operand> evaluates to <initial>. This is essentially equivalent to JavaScript's Array.reduce function.

All other special array operations can be implemented using (only) a reduce operation with a suitable <lambda>.

To be able to access values in the original data context, CertLogic may expand beyond JsonLogic at some point by also adding a key-value pair with key "data" to the data object passed to the <lambda>, whose value is the original data context.

Appendix D - multiple events checking

This is currently not in scope.

Multiple scans (for information only, and for discussion)

- At the EU level, only 1 QR code is to be scanned. At the MS level, scenarios can be worked out where more than 1 QR code must be scanned. Examples:
  - The requirement to show two Tests, with a certain interval
  - A person that has had COVID-19 needs only 1 vaccination dosis
- For these use cases, and for maximum flexibility, a methodology called "Two Points" can be used:
- FFT is reached when at least 2 points have been reached. The following rules apply:
- At the beginning of the FFT check, the FFT status is 0.
- MS decide on the weight of V, T and R for their acceptance rules. These can be 2, 1 or 0 or 4. OR: V/R/T = 2
- V: divide "nr of vaccinations" by "nr of complete course" and multiply by 2. Example: "1/2" = 1 point, "1/1" or "2/2" = 2 points.
- T: If the weight = 1, then two tests are needed for a FFT status. If it is 2, then 1 test is enough.
- R: If the weight = 1, then another proof is needed to reach the 2 points. If it is 2, the recovery statement is enough for the FFT.

To do list

- Add Rule Container format for Gateway
- Add Valueset Description for Gateway
- Add Wallet/Verifier Architecture Guide
- Add Checklist Design
- Add Architecture Proposals for API
- Add JSON Logic Definitions/Requirements