Logius Standard - Proposed version

NL GOV profile for CloudEvents 1.1

Notificatieservices

Logius Standard
Proposed version January 07, 2026

This version:
https://gitdocumentatie.logius.nl/publicatie/notificatieservices/cloudevents-nl/1.1/

Latest published version:
https://gitdocumentatie.logius.nl/publicatie/notificatieservices/cloudevents-nl/

Latest editor's draft:
https://logius-standaarden.github.io/NL-GOV-profile-for-CloudEvents/

Previous version:
https://gitdocumentatie.logius.nl/publicatie/notificatieservices/cloudevents-nl/1.0/

Editors:
Alexander Green (Logius)
Stas Mironov (Logius)

Authors:
Ad Gerrits (VNG Realisatie)
Gershon Jansen (VNG Realisatie)
Jeanot Bijpost (VNG Realisatie)

Participate:
GitHub Logius-standaarden/NL-GOV-profile-for-CloudEvents
File an issue
Commit history
Pull requests

This document is also available in these non-normative format: PDF

Status of This Document

This is the definitive concept of this document. Edits resulting from consultations have been applied.

Table of Contents

§ Conformance

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

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

This document is an adaptation of the CloudEvents specification for describing event data in common formats to provide interoperability across services, platforms and systems. This does not indicate an endorsement by the CloudEvents working group. As far as the specification is incorporated in this document, the CloudEvents License applies.

> NOTE
>
> Het Nederlandse profiel op Cloudevents is als standaard in beheer bij Logius.
>
> Het profiel is ontwikkeld door VNG in opdracht van BZK. Ontwikkeling van het profiel is als project afgesloten medio 2022 waarna het door VNG is overgedragen om in beheer genomen te worden.

Abstract

Abstract

CloudEvents is a vendor neutral specification for defining the format of event data. This specification profiles the CloudEvents specification to standardize the automated exchange of information about events, specifically applicable to the Dutch government. The Governance of this standard is described by the API-Standaarden beheermodel published by Logius.

§ Dutch government profile for CloudEvents

This profile is based on *CloudEvents - Version 1.0.1* as published by the Serverless Working Group of the Cloud Native Computing Foundation. It should be considered a fork of this profile as the CloudEvents specification is geared more towards generic use and in the Netherlands we want to add a number of requirements for the Dutch situation with the goal to agree on how to use the CloudEvents specification.

The goal of the CloudEvents specification is to define interoperability of event systems that allow services to produce or consume events, where the producer and consumer can be developed and deployed independently. The ability to keep services loosely coupled within a distributed system such as the Dutch government makes it possible to work more often and in a better event-driven way. Using the CE standard supports this and makes maximum use of existing worldwide standards.

The CloudEvents standard is based on the principle of not imposing more requirements on the parties involved than necessary. This means, among other things, that there are no requirements for how consumers should interpret and process received notifications. Constraints pertaining to consumers are therefore more often formulated with '*MAY*' than with '*SHOULD*' or '*MUST*' (e.g. "Consumers *MAY* assume that Events with identical `source` and `id` are duplicates")) The GOV NL profile endorses this principle. In practice, the parties involved are of course allowed to apply stricter constraints.

Starting with chapter Introduction we follow the structure of the CloudEvents profile. Where we do not use content from CloudEvents we use ~~strikethrough~~ to indicate it is not part of CloudEvents-NL. Where we have added more specific requirements for the Dutch situation this is indicated with the **CloudEvents-NL** tag.

§ Usecases

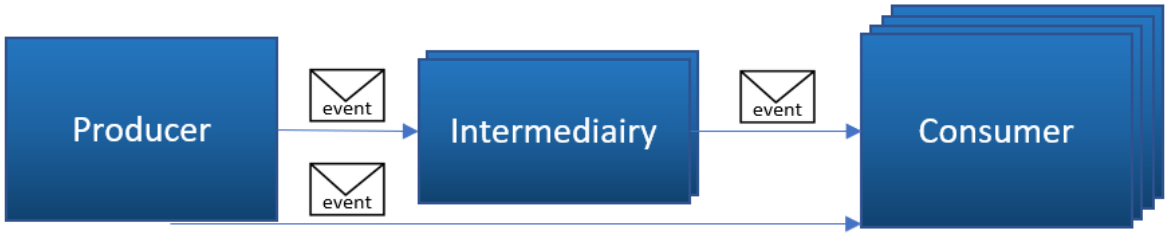

Figure 1 Publish-subscribe pattern

§ Introduction

The basic pattern for use cases describes a (public/governmental) application in the role of 'producer' that publishes 'events': data records expressing an occurrence and its context. Published events can be consumed by applications in the role of 'consumer'. Consumers subscribe to certain types of events. There may be one or more applications in the role of 'intermediary' that take care of routing events to consumers based on contextual information. This is akin to the publish-subscribe pattern.

Within this context, it concerns standardization of the automated exchange of event information via applications. In practice, agreements at business level are often also required between the parties involved.

1. Overview

Events are everywhere. However, event producers tend to describe events differently.

The lack of a common way of describing events means developers are constantly re-learning how to consume events. This also limits the potential for libraries, tooling and infrastructure to aid the delivery of event data across environments, like SDKs, event routers or tracing systems. The portability and productivity that can be achieved from event data is hindered overall.

CloudEvents is a specification for describing event data in common formats to provide interoperability across services, platforms and systems.

Event Formats specify how to serialize a CloudEvent with certain encoding formats. Compliant CloudEvents implementations that support those encodings *MUST* adhere to the encoding rules specified in the respective event format. All implementations *MUST* support the JSON format.

For more information on the history, development and design rationale behind the specification, see the CloudEvents Primer document.

2. Notations and Terminology

2.1 Notational Conventions

The key words "*MUST*", "*MUST NOT*", "*REQUIRED*", "*SHALL*", "*SHALL NOT*", "*SHOULD*", "*SHOULD NOT*", "*RECOMMENDED*", "*MAY*", and "*OPTIONAL*" in this document are to be interpreted as described in RFC 2119.

For clarity, when a feature is marked as "*OPTIONAL*" this means that it is *OPTIONAL* for both the Producer and Consumer of a message to support that feature. In other words, a producer can choose to include that feature in a message if it wants, and a consumer can choose to support that feature if it wants. A consumer that does not support that feature will then silently ignore that part of the message. The producer needs to be prepared for the situation where a consumer ignores that feature. An Intermediary *SHOULD* forward *OPTIONAL* attributes.

2.2 Terminology

This specification defines the following terms:

§ 2.2.1 Data Schema

The "dataschema" refers to the technical specification of the attributes in a given event. Currently v1.0.2 of CloudEvents specifies various different formats for documenting the dataschema.

"Dataschema" attribute is expected to be informational, largely to be used during development and by tooling that is able to provide diagnostic information over arbitrary CloudEvents with a data content type understood by that tooling. For further specifications, see 3.4.2 dataschema.]

§ 2.2.2 Occurrence

An "occurrence" is the capture of a statement of fact during the operation of a software system. This might occur because of a signal raised by the system or a signal being observed by the system, because of a state change, because of a timer elapsing, or any other noteworthy activity. For example, a device might go into an alert state because the battery is low, or a virtual machine is about to perform a scheduled reboot.

§ 2.2.3 Event

An "event" is a data record expressing an occurrence and its context. Events are routed from an event producer (the source) to interested event consumers. The routing can be performed based on information contained in the event, but an event will not identify a specific routing destination. Events will contain two types of information: the Event Data representing the Occurrence and Context metadata providing contextual information about the Occurrence. A single occurrence *MAY* result in more than one event.

§ 2.2.4 Producer

The "producer" is a specific instance, process or device that creates the data structure describing the CloudEvent.

§ 2.2.5 Source

The "source" is the context in which the occurrence happened. In a distributed system it might consist of multiple Producers. If a source is not aware of CloudEvents, an external producer creates the CloudEvent on behalf of the source.

§ 2.2.6 Consumer

A "consumer" receives the event and acts upon it. It uses the context and data to execute some logic, which might lead to the occurrence of new events.

§ 2.2.7 Intermediary

An "intermediary" receives a message containing an event for the purpose of forwarding it to the next receiver, which might be another intermediary or a Consumer. A typical task for an intermediary is to route the event to receivers based on the information in the Context.

§ 2.2.8 Context

Context metadata will be encapsulated in the Context Attributes. Tools and application code can use this information to identify the relationship of Events to aspects of the system or to other Events.

§ 2.2.9 Data

Domain-specific information about the occurrence (i.e. the payload). This might include information about the occurrence, details about the data that was changed, or more. See the Event Data section for more information.

§ 2.2.10 Event Format

An Event Format specifies how to serialize a CloudEvent as a sequence of bytes. Stand-alone event formats, such as the JSON format, specify serialization independent of any protocol or storage medium. Protocol Bindings *MAY* define formats that are dependent on the protocol.

§ 2.2.11 Message

Events are transported from a source to a destination via messages.

A "structured-mode message" is one where the event is fully encoded using a stand-alone event format and stored in the message body.

A "binary-mode message" is one where the event data is stored in the message body, and event attributes are stored as part of message meta-data.

§ 2.2.12 Protocol

Messages can be delivered through various industry standard protocol (e.g. HTTP, AMQP, MQTT, SMTP), open-source protocols (e.g. Kafka, NATS), or platform/vendor specific protocols (AWS Kinesis, Azure Event Grid).

§ 2.2.13 Protocol Binding

A protocol binding describes how events are sent and received over a given protocol.

Protocol bindings *MAY* choose to use an Event Format to map an event directly to the transport envelope body, or *MAY* provide additional formatting and structure to the envelope. For example, a wrapper around a structured-mode message might be used, or several messages could be batched together into a transport envelope body.

3. Context Attributes

Every CloudEvent conforming to this specification *MUST* include context attributes designated as *REQUIRED*, *MAY* include one or more *OPTIONAL* context attributes and *MAY* include one or more extension attributes.

These attributes, while descriptive of the event, are designed such that they can be serialized independent of the event data. This allows for them to be inspected at the destination without having to deserialize the event data.

3.1 Attribute Naming Convention

The CloudEvents specifications define mappings to various protocols and encodings, and the accompanying CloudEvents SDK targets various runtimes and languages. Some of these treat metadata elements as case-sensitive while others do not, and a single CloudEvent might be routed via multiple hops that involve a mix of protocols, encodings, and runtimes. Therefore, this specification limits the available character set of all attributes such that case-sensitivity issues or clashes with the permissible character set for identifiers in common languages are prevented.

CloudEvents attribute names *MUST* consist of lower-case letters ('a' to 'z') or digits ('0' to '9') from the ASCII character set. Attribute names *SHOULD* be descriptive and terse and *SHOULD NOT* exceed 20 characters in length.

3.2 Type System

The following abstract data types are available for use in attributes. Each of these types *MAY* be represented differently by different event formats and in protocol metadata fields. This specification defines a canonical string-encoding for each type that *MUST* be supported by all implementations.

- `Boolean` - a boolean value of "true" or "false".
  - String encoding: a case-sensitive value of `true` or `false`.
- `Integer` - A whole number in the range -2,147,483,648 to +2,147,483,647 inclusive. This is the range of a signed, 32-bit, twos-complement encoding. Event formats do not have to use this encoding, but they *MUST* only use `Integer` values in this range.

    - String encoding: Integer portion of the JSON Number per RFC 7159, Section 6
- `String` - Sequence of allowable Unicode characters. The following characters are disallowed:
    - the "control characters" in the ranges U+0000-U+001F and U+007F-U+009F (both ranges inclusive), since most have no agreed-on meaning, and some, such as U+000A (newline), are not usable in contexts such as HTTP headers.
    - code points identified as noncharacters by Unicode.
    - code points identifying Surrogates, U+D800-U+DBFF and U+DC00-U+DFFF, both ranges inclusive, unless used properly in pairs. Thus (in JSON notation) "\uDEAD" is invalid because it is an unpaired surrogate, while "\uD800\uDEAD" would be legal.
- `Binary` - Sequence of bytes.
    - String encoding: Base64 encoding per RFC4648.
- `URI` - Absolute uniform resource identifier.
    - String encoding: `Absolute URI` as defined in RFC 3986 Section 4.3.
- `URI-reference` - Uniform resource identifier reference.
    - String encoding: `URI-reference` as defined in RFC 3986 Section 4.1.
- `Timestamp` - Date and time expression using the Gregorian Calendar.
    - String encoding: RFC 3339.

All context attribute values *MUST* be of one of the types listed above. Attribute values *MAY* be presented as native types or canonical strings.

A strongly-typed programming model that represents a CloudEvent or any extension *MUST* be able to convert from and to the canonical string-encoding to the runtime/language native type that best corresponds to the abstract type.

For example, the `time` attribute might be represented by the language's native *datetime* type in a given implementation, but it *MUST* be settable providing an RFC3339 string, and it *MUST* be convertible to an RFC3339 string when mapped to a header of an HTTP message.

A CloudEvents protocol binding or event format implementation *MUST* likewise be able to convert from and to the canonical string-encoding to the corresponding data type in the encoding or in protocol metadata fields.

An attribute value of type `Timestamp` might indeed be routed as a string through multiple hops and only materialize as a native runtime/language type at the producer and ultimate consumer. The `Timestamp` might also be routed as a native protocol type and might be mapped to/from the respective language/runtime types at the producer and consumer ends, and never materialize as a string.

The choice of serialization mechanism will determine how the context attributes and the event data will be serialized. For example, in the case of a JSON serialization, the context attributes and the event data might both appear within the same JSON object.

§ 3.3 *REQUIRED* Attributes

The following attributes are *REQUIRED* to be present in all CloudEvents:

§ 3.3.1 id

- Type: `String`
- Description: Identifies the event. Producers *MUST* ensure that `source` + `id` is unique for each distinct event. ~~If a duplicate event is re-sent (e.g. due to a network error) it *MAY* have the same `id`.~~ Consumers *MAY* assume that Events with identical `source` and `id` are duplicates.
- Constraints:
  - *REQUIRED*
  - *MUST* be a non-empty string
  - *MUST* be unique within the scope of the producer
- Examples:
  - An ID counter maintained by the producer
  - A UUID

CloudEvents-NL: Additional content

- Constraints:
  - If an ID is available that can persistently identify the event, producers *MUST* use that ID. For example so that consumers may use `id` to request information about the event from the source.
  - If no ID is available that can persistently identify the event producers *SHOULD* use a random ID:
    - *SHOULD* use a UUID.
    - *MUST* describe the limitations (e.g., that it's just a random ID and has no relation to the occurrence event) in the 3.4.2 dataschema attribute.
- Examples:
  - 'doc2021033441' (ID of the document created as a result of an event that occurred).
  - 'f3dce042-cd6e-4977-844d-05be8dce7cea' (UUID generated with the sole function of being able to uniquely identify the event.

§ 3.3.2 source

- Type: `URI-reference`
- Description: Identifies the context in which an event happened. Often this will include information such as the type of the event source, the organization publishing the event or the process that produced the event. The exact syntax and semantics behind the data encoded in the URI is defined by the event producer.

  Producers *MUST* ensure that `source` + `id` is unique for each distinct event.

  An application *MAY* assign a unique `source` to each distinct producer, which makes it easy to produce unique IDs since no other producer will have the same source. The application *MAY* use UUIDs, URNs, DNS authorities or an application-specific scheme to create unique `source` identifiers.

  A source *MAY* include more than one producer. In that case the producers *MUST* collaborate to ensure that `source` + `id` is unique for each distinct event.
- Constraints:
  - *REQUIRED*
  - *MUST* be a non-empty URI-reference

    - An absolute URI is *RECOMMENDED*
- Examples
    - Internet-wide unique URI with a DNS authority.
        - https://github.com/cloudevents
        - mailto:cncf-wg-serverless@lists.cncf.io
    - Universally-unique URN with a UUID:
        - urn:uuid:6e8bc430-9c3a-11d9-9669-0800200c9a66
    - Application-specific identifiers
        - /cloudevents/spec/pull/123
        - /sensors/tn-1234567/alerts
        - 1-555-123-4567

CloudEvents-NL: Additional content

- Constraints:
  - *SHOULD* be a URN notation with 'nld' as namespace identifier.
  - *SHOULD* contain consecutive a unique identifier of:
    - the organization that publishes the event
    - the source system that publishes the event.
  - involved organizations *SHOULD* agree on how organizations and systems are uniquely identified (e.g. via the use of OIN, KVK-nummer or for organization identification);
    - In line with *API Design Rules*:
      - *SHOULD* use the "organisatie-identificatienummer" (OIN) for identifying Dutch government organizations
      - *SHOULD* use the KvK-nummer for identifying Dutch non-government organizations (companies, associations, foundations etc...)
      - *SHOULD* use the eIDAS legal identifier in the EU context. national, European or worldwide)
    - *SHOULD* choose an abstraction level for the source that can be used sustainably; even if the initial scope expands (e.g., scope creep from domain specific to more general categorization).
  - *MUST NOT* be used to reference an external data location (see 3.4.7 dataref).
- Examples:
  - urn:nld:oin:00000001823288444000:systeem:BRP-component
  - urn:nld:kvknr:09220932.burgerzakensysteem
  - urn:nld:gemeente-nijmegen.burgerzakensysteem
  - urn:nld:gemeente-Bergen%20%28L%29.burgerzakensysteem ***Comment***: The use of (unique) descriptions increases recognisability, but also has disadvantages such as occurred changes or required encoding (like in the above example where "Bergen (L)" requires encoding).

§ 3.3.3 specversion

- Type: `String`

- Description: The version of the CloudEvents specification which the event uses. This enables the interpretation of the context. Compliant event producers *MUST* use a value of `1.0` when referring to this version of the specification.

  Currently, this attribute will only have the 'major' and 'minor' version numbers included in it. This allows for 'patch' changes to the specification to be made without changing this property's value in the serialization. Note: for 'release candidate' releases a suffix might be used for testing purposes.

- Constraints:
  - *REQUIRED*
  - *MUST* be a non-empty string

§ 3.3.4 type

- Type: `String`
- Description: This attribute contains a value describing the type of event related to the originating occurrence. Often this attribute is used for routing, observability, policy enforcement, etc. The format of this is producer defined and might include information such as the version of the `type` - see Versioning of CloudEvents in the Primer for more information.
- Constraints:
  - *REQUIRED*
  - *MUST* be a non-empty string
  - *SHOULD* be prefixed with a reverse-DNS name. The prefixed domain dictates the organization which defines the semantics of this event type.
- Examples
  - com.github.pull_request.opened
  - com.example.object.deleted.v2

CloudEvents-NL: Additional content

Constraints:

- *MUST* be Reverse domain name notation
- *MAY* be further specified by adding a suffix (for example: `nl.brp.verhuizing.binnengemeentelijk` instead of `nl.brp.binnengemeentelijke-verhuizing`)
- Producers *MUST* facilitate consumers to request additional information on the type and adequatly explain the exact meaning.
- *SHOULD* stay the same when a CloudEvent's data changes in a backwardly-compatible way.
- *SHOULD* change when a CloudEvent's data changes in a backwardly-incompatible way.
- The producer *SHOULD* produce both the old event and the new event for some time (potentially forever) in order to avoid disrupting consumers.
- The producer decides if versioning is used.
- If versioning is used, the type attribute *MUST* only include a single version number, prefixed by the letter `v`
- In descending order of preference one *SHOULD* use the name of a:
  - data source (for example: 'nl.brp.persoon-verhuisd)
  - domain (for example: nl.natuurlijke-personen.persoon-verhuisd); for domain designation plural *MUST* be used.
  - law or rule (for example: nl.amsterdam.erfpacht.overdracht)
- Names of organizations *SHOULD NOT* be used (because they are not time invariant).

§ 3.4 *OPTIONAL* Attributes

The following attributes are *OPTIONAL* to appear in CloudEvents. See the Notational Conventions section for more information on the definition of *OPTIONAL*.

§ 3.4.1 datacontenttype

- Type: `String` per RFC 2046

- Description: Content type of `data` value. This attribute enables `data` to carry any type of content, whereby format and encoding might differ from that of the chosen event format. For example, an event rendered using the JSON envelope format might carry an XML payload in `data`, and the consumer is informed by this attribute being set to "application/xml". The rules for how `data` content is rendered for different `datacontenttype` values are defined in the event format specifications; for example, the JSON event format defines the relationship in section 3.1.

  For some binary mode protocol bindings, this field is directly mapped to the respective protocol's content-type metadata property. Normative rules for the binary mode and the content-type metadata mapping can be found in the respective protocol

  In some event formats the `datacontenttype` attribute *MAY* be omitted. For example, if a JSON format event has no `datacontenttype` attribute, then it is implied that the `data` is a JSON value conforming to the "application/json" media type. In other words: a JSON-format event with no `datacontenttype` is exactly equivalent to one with `datacontenttype="application/json"`.

  When translating an event message with no `datacontenttype` attribute to a different format or protocol binding, the target `datacontenttype` *SHOULD* be set explicitly to the implied `datacontenttype` of the source.

- Constraints:
  - *OPTIONAL*
  - If present, *MUST* adhere to the format specified in RFC 2046
- For Media Type examples see IANA Media Types

§ *3.4.1.1 CloudEvents-NL*

Constraints:

- JSON-format *SHOULD* be used (see *API Design Rules*). Part of this is the intention to name JSON as the primary representation format for APIs. Because APIs play an important role in communicating events (e.g., when using the webhook pattern) the JSON format is preferred to use for payload data).

§ 3.4.2 dataschema

- Type: `URI`
- Description: Identifies the schema that `data` adheres to. Incompatible changes to the schema *SHOULD* be reflected by a different URI. See Versioning of CloudEvents in the Primer for more information.
- Constraints:
  - *OPTIONAL*
  - If present, *MUST* be a non-empty URI

§ *3.4.2.1 CloudEvents-NL*

Constraints:

- It *SHOULD* be prevented that different schedules arise for the same data.
- The dataschema attribute is expected to be informational, largely to be used during development and by tooling that is able to provide diagnostic information over arbitrary CloudEvents with a data content type understood by that tooling (see: The role of the dataschema attribute within versioning

§ 3.4.3 subject

- Type: `String`
- Description: This describes the subject of the event in the context of the event producer (identified by `source`). In publish-subscribe scenarios, a subscriber will typically subscribe to events emitted by a `source`, but the `source` identifier alone might not be sufficient as a qualifier for any specific event if the `source` context has internal sub-structure.

  Identifying the subject of the event in context metadata (opposed to only in the `data` payload) is particularly helpful in generic subscription filtering scenarios where middleware is unable to interpret the `data` content. In the above example, the subscriber might only be interested in blobs with names ending with '.jpg' or '.jpeg' and the `subject` attribute allows for constructing a simple and efficient string-suffix filter for that subset of events.

- Constraints:
  - *OPTIONAL*
  - If present, *MUST* be a non-empty string
- Example:
  - A subscriber might register interest for when new blobs are created inside a blob-storage container. In this case, the event `source` identifies the subscription scope (storage container), the `type` identifies the "blob created" event, and the `id` uniquely identifies the event instance to distinguish separate occurrences of a same-named blob having been created; the name of the newly created blob is carried in `subject`:
    - `source: https://example.com/storage/tenant/container`
    - `subject: mynewfile.jpg`

§ *3.4.3.1 CloudEvents-NL*

Constraints:

- Decision on whether or not to use the attribute and/or the exact interpretation is postponed. To be determined partly on the basis of future agreements about subscription and filtering.

Example:

- `source: urn:nld:oin:00000001823288444000:systeem:BRP-component`
- `type: nl.brp.persoon-gehuwd`
- `subject: 999990342` (citizen service number)

§ **3.4.4 time**

- Type: `Timestamp`
- Description: Timestamp of when the occurrence happened. If the time of the occurrence cannot be determined then this attribute *MAY* be set to some other time (such as the current time) by the CloudEvents producer, however all producers for the same `source` *MUST* be consistent in this respect. In other words, either they all use the actual time of the occurrence or they all use the same algorithm to determine the value used.
- Constraints:
  - *OPTIONAL*

- If present, *MUST* adhere to the format specified in RFC 3339

§ *3.4.4.1 CloudEvents-NL*

- The time the event was logged *SHOULD* be used (in many cases this is the only time that can be determined unambiguously).
- The exact meaning of `time` *MUST* be clearly documented.
- The time when an event occurred in reality *SHOULD NOT* be used (if there is a need for this among consumers, this can be included in payload data).
- If the time when an event occurred in reality is needed for things like routing or filtering, it can be included as a context attribute by the producer.

§ **3.4.5 Extension Context Attributes**

A CloudEvent *MAY* include any number of additional context attributes with distinct names, known as "extension attributes". Extension attributes *MUST* follow the same naming convention and use the same type system as standard attributes. Extension attributes have no defined meaning in this specification, they allow external systems to attach metadata to an event, much like HTTP custom headers.

Extension attributes are always serialized according to binding rules like standard attributes. However this specification does not prevent an extension from copying event attribute values to other parts of a message, in order to interact with non-CloudEvents systems that also process the message. Extension specifications that do this *SHOULD* specify how receivers are to interpret messages if the copied values differ from the cloud-event serialized values.

§ **3.4.6 Defining Extensions**

See CloudEvent Attributes Extensions for additional information concerning the use and definition of extensions.

The definition of an extension *SHOULD* fully define all aspects of the attribute - e.g. its name, type, semantic meaning and possible values. New extension definitions *SHOULD* use a name that is descriptive enough to reduce the chances of name collisions with other extensions. In particular, extension authors *SHOULD* check the documented extensions document for the set of known extensions - not just for possible name conflicts but for extensions that might be of interest.

Many protocols support the ability for senders to include additional metadata, for example as HTTP headers. While a CloudEvents receiver is not mandated to process and pass them along, it is *RECOMMENDED* that they do so via some mechanism that makes it clear they are non-CloudEvents metadata.

Here is an example that illustrates the need for additional attributes. In many IoT and enterprise use cases, an event could be used in a serverless application that performs actions across multiple types of events. To support such use cases, the event producer will need to add additional identity attributes to the "context attributes" which the event consumers can use to correlate this event with the other events. If such identity attributes happen to be part of the event "data", the event producer would also add the identity attributes to the "context attributes" so that event consumers can easily access this information without needing to decode and examine the event data. Such identity attributes can also be used to help intermediate gateways determine how to route the events.

§ *3.4.6.1 CloudEvents-NL*

- Two of the extension attributes included by CloudEvents ('dataref' and 'sequence') are included as optional attributes in the CloudEvents-NL profile because it is foreseen that there is often a need to use these attributes.
- Extension attributes should be kept minimal to ensure the CloudEvent can be properly serialized and transported (e.g. when using HTTP-headers most HTTP servers will reject requests with excessive HTTP header data).

§ **3.4.7 dataref**

- Type: `URI-reference`
- Description: A reference to a location where the event payload is stored. The location *MAY* not be accessible without further information (e.g. a pre-shared secret).

  Known as the "Claim Check Pattern", this attribute *MAY* be used for a variety of purposes, including:

  - If the Data is too large to be included in the message, the `data` is not present, and the consumer can retrieve it using this attribute.
  - If the consumer wants to verify that the Data has not been tampered with, it can retrieve it from a trusted source using this attribute.

    - If the Data *MUST* only be viewed by trusted consumers (e.g. personally identifiable information), only a trusted consumer can retrieve it using this attribute and a pre-shared secret.

  If this attribute is used, the information *SHOULD* be accessible long enough for all consumers to retrieve it, but *MAY* not be stored for an extended period of time.

- Constraints:
    - *OPTIONAL*

§ *3.4.7.1 Example*

The following example shows a CloudEvent in which the event producer has included both `data` and `dataref` (serialized as JSON):

```
{
    "specversion" : "1.0",
    "type" : "com.github.pull_request.opened",
    "source" : "https://github.com/cloudevents/spec/pull/123",
    "id" : "A234-1234-1234",
    "datacontenttype" : "text/xml",
    "data" : "<much wow=\"xml\"/>",
    "dataref" : "https://github.com/cloudevents/spec/pull/123/events/A234
}
```

§ *3.4.7.2 CloudEvents-NL*

- *MAY* be used to reference an external data location (for example: a link back to the producer of the event that can be queried for more information about the event).
- *MAY* be used to implenment 'informatiearm notificeren' where the consumer of the event receives some minimal information on the nature of the event, but then has to issue a request back to the producer to obtain additional information (the time aspect may deserve attention because changes may occur in the period that consumers are notified and the time of requesting additional information).

§ 3.5 Sequence

This extension defines two attributes that can be included within a CloudEvent to describe the position of an event in the ordered sequence of events produced by a unique event source. The `sequence` attribute represents the value of this event's order in the stream of events. The exact value and meaning of this attribute is defined by the `sequencetype` attribute. If the `sequencetype` is missing, or not defined in this specification, event consumers will need to have some out-of-band communication with the event producer to understand how to interpret the value of the attribute.

§ 3.5.1 Attributes

§ *3.5.1.1 sequence*

- Type: `String`
- Description: Value expressing the relative order of the event. This enables interpretation of data supercedence.
- Constraints
  - *REQUIRED*
  - *MUST* be a non-empty lexicographically-orderable string
  - *RECOMMENDED* as monotonically increasing and contiguous

§ *3.5.1.2 sequencetype*

- Type: `String`
- Description: Specifies the semantics of the sequence attribute. See the SequenceType Values section for more information.
- Constraints:
  - *OPTIONAL*
  - If present, *MUST* be a non-empty string

§ 3.5.1.2.1 SequenceType Values

This specification defines the following values for `sequencetype`. Additional values *MAY* be defined by other specifications.

§ 3.5.1.2.2 Integer

If the `sequencetype` is set to `Integer`, the `sequence` attribute has the following semantics:

- The values of `sequence` are string-encoded signed 32-bit Integers.
- The sequence *MUST* start with a value of `1` and increase by `1` for each subsequent value (i.e. be contiguous and monotonically increasing).
- The sequence wraps around from 2,147,483,647 (2^31 -1) to -2,147,483,648 (-2^31).

§ *3.5.1.3 CloudEvents-NL*

- Attribute 'sequence' can be helpful in situations where:
- a form of 'pull mechanism' is used ((e.g. periodically fetching events by consumers via HTTP request)) or
- where there is a need for (re)synchronization (e.g. after errors have occurred).

§ 4. Event Data

As defined by the term Data, CloudEvents *MAY* include domain-specific information about the occurrence. When present, this information will be encapsulated within `data`.

- Description: The event payload. This specification does not place any restriction on the type of this information. It is encoded into a media format which is specified by the `datacontenttype` attribute (e.g. application/json), and adheres to the `dataschema` format when those respective attributes are present.
- Constraints:
  - *OPTIONAL*

5. Size Limits

In many scenarios, CloudEvents will be forwarded through one or more generic intermediaries, each of which might impose limits on the size of forwarded events. CloudEvents might also be routed to consumers, like embedded devices, that are storage or memory-constrained and therefore would struggle with large singular events.

The "size" of an event is its wire-size and includes every bit that is transmitted on the wire for the event: protocol frame-metadata, event metadata, and event data, based on the chosen event format and the chosen protocol binding.

If an application configuration requires for events to be routed across different protocols or for events to be re-encoded, the least efficient protocol and encoding used by the application *SHOULD* be considered for compliance with these size constraints:

- Intermediaries *MUST* forward events of a size of 64 KByte or less.
- Consumers *SHOULD* accept events of a size of at least 64 KByte.

In effect, these rules will allow producers to publish events up to 64KB in size safely. Safely here means that it is generally reasonable to expect the event to be accepted and retransmitted by all intermediaries. It is in any particular consumer's control, whether it wants to accept or reject events of that size due to local considerations.

Generally, CloudEvents publishers *SHOULD* keep events compact by avoiding embedding large data items into event payloads and rather use the event payload to link to such data items. From an access control perspective, this approach also allows for a broader distribution of events, because accessing event-related details through resolving links allows for differentiated access control and selective disclosure, rather than having sensitive details embedded in the event directly.

6. Advanced CloudEvents Security Options

Interoperability is the primary driver behind this specification, enabling such behavior requires some information to be made available *in the clear* resulting in the potential for information leakage.

Consider the following to prevent inadvertent leakage especially when leveraging 3rd party platforms and communication networks:

- Context Attributes

Sensitive information *SHOULD NOT* be carried or represented in context attributes.

CloudEvent producers, consumers, and intermediaries *MAY* introspect and log context attributes.

- Data

  Domain specific event data *SHOULD* be encrypted to restrict visibility to trusted parties. The mechanism employed for such encryption is an agreement between producers and consumers and thus outside the scope of this specification.

- Protocol Bindings

  Protocol level security *SHOULD* be employed to ensure the trusted and secure exchange of CloudEvents.

§ 7. Example

The following example shows a CloudEvent serialized as JSON:

```
{
    "specversion" : "1.0",
    "type" : "com.github.pull_request.opened",
    "source" : "https://github.com/cloudevents/spec/pull",
    "subject" : "123",
    "id" : "A234-1234-1234",
    "time" : "2018-04-05T17:31:00Z",
    "comexampleextension1" : "value",
    "comexampleothervalue" : 5,
    "datacontenttype" : "text/xml",
    "data" : "<much wow=\"xml\"/>"
}
```

CloudEvents-NL

In the example below, a number of agreements are visible as they apply within the CloudEvents-NL profile. In order to show as much things as possible, this is done in the form of a very extensive message. In minimal form, a message contains only four mandatory attributes: `id`, `source`, `specversion` and `type`. For more information about a particular attribute, see the detailed attribute description.

```
{
  "specversion": "1.0",
  "type": "nl.overheid.zaken.zaakstatus-gewijzigd",
```

```
  "source": "urn:nld:oin:00000001823288444000:systeem:BRP-component",
  "subject": "999990342",
  "id": "f3dce042-cd6e-4977-844d-05be8dce7cea",
  "time": "2021-12-10T17:31:00Z",
  "nlbrpnationaliteit": "0083",
  "geheimnummer": null,
  "dataref": "https://gemeenteX/api/persoon/999990342",
  "sequence": "1234",
  "sequencetype": "integer",
  "datacontenttype": "application/json",
  "data": {
    "bsn": "999990342",
    "naam": "Jan Jansen",
    "gecontroleerd": "ja"
  }
}
```


Attribute	Explanation
specversion	Always '1.0'
type	Reverse DNS notation
source	Urn notation with 'nld' namespace identifier
subject	BSN as example; yet to be seen if and how attribute 'subject' will be used
id	Uuid (unique)
time	Time when event was recorded
nlbrpnationaliteit	Extension attribute with a BRP-domain specific meaning
geheimnummer	Extension attribute, to be treated as the equivalent of unset or omitted
dataref	Extension attribute with a reference to where to get additional information
sequence	Extension attribute with an event tracking number
sequencetype	Extension attribut with indication of the type of sequence used
datacontenttype	Indication of content type in attribute 'data'
data	Content information for consumer ('payload')

Attribute names meet the CloudEvents specification requirements:

- *MUST* consist of lower-case letters ('a' to 'z') or digits ('0' to '9') from the ASCII character set.
- *SHOULD* be descriptive and terse and *SHOULD NOT* exceed 20 characters in length.

§ A. Use of JSON, HTTP and Webhook

This appendix describes how the CloudEvent-NL profile can be applied when using the JSON format, the HTTP protocol and the Webhook pattern.

The CloudEvent-NL message format can be used when using different formats and protocols. CloudEvents has a layered architecture for this. In order to be able to use the GOV-NL profile properly in practice, agreements must also be made when a certain format and/or protocol is used.

In addition to the CloudEvent specification the Serverless Working Group has described for several commonly used formats and protocols how they can be used in a standardized way in combination with the CloudEvents message format.

Within the Dutch government, frequent use is made of the JSON format, the HTTP protocol and the Webhook pattern for data exchange. For example, a common way to send events to consumers is to use the webhook pattern where a message in JSON format is sent via the HTTP protocol. Further standardization than just event description via the NL GOV profile therefore benefits most from agreements around these 3 areas. In addition to standardization through the use of the NL GOV profile, we therefore work towards standardization on exchanging event information when using JSON, HTTP and Webhook.

The NL GOV profile is intended to be used as a government-wide standard. This does not yet apply to the additional specification for the use of JSON, HTTP and Webhook. The specifications for them have the character of 'guidelines' ("a statement by which to determine a course of action", Wikipedia).

Similar to what happened in the NL GOV profile for the CloudEvents specification, the guidelines make recommendations about the use of the specifications within the context of the Dutch government. These are intended to make use of the specifications more unambiguous and to work towards standardisation in the long term.

For the time being, the following constraints apply:

- One *SHOULD* use the JSON Event Format for CloudEvents specification and pay attention to the points of attention and recommendations in the guideline NL GOV Guideline for CloudEvents JSON.
- One *SHOULD* use the HTTP Protocol Binding for CloudEvents specification and pay attention to the points of attention and recommendations in the guideline NL GOV Guideline for CloudEvents HTTP.
- One *SHOULD* use the HTTP 1.1 Web Hooks for Event Delivery specification and pay attention to the points of attention and recommendations in the guideline NL GOV Guideline

§ B. List of Figures

§ C. References

§ C.1 Normative references

[ADR]
API Design Rules. Jasper Roes; Joost Farla. Logius. URL: https://gitdocumentatie.logius.nl/publicatie/api/adr/2.1

[CloudEvents]
CloudEvents - Version 1.0.1. CNCF Serverless Working Group. May 2011. URL: https://github.com/cloudevents/spec/blob/v1.0.1/spec.md

[RFC2119]
Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc2119

[RFC8174]
Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc8174

↥