2023.06.09
Draft Version 1.0.0

Interoperable Dialog Event Object Specification Version 1.0

The Open Voice Network
Architecture Work Group of the Technical Committee

Editor-in-chief: David Attwater
Contributors: Emmett Coin
Deborah Dahl
Jim Larson

June 9, 2023

TABLE OF CONTENTS

Chapter 0. Scope and Introduction

0.1 Document Scope

This document specifies the format for Open Voice Network (OVON) interoperable dialog events. The requirements for this specification are given in Interoperable Dialog Packet Requirements [10]. This specification is generic. As described in the requirements there are many different potential uses of a dialog event.

0.2 Dialog Events

Interoperable conversational systems will need to be able to process linguistic input and generate linguistic output. The components within such systems also need to send and receive such output in a standardized way.

The purpose of a dialog event is to define a generic standardized data structure that can be used in any component of a dialog system to express a 'language event', that is to say, any features associated with a phrase, utterance or part of an utterance. Dialog events span a certain time period and are associated with a single speaker.

These events are used to represent user inputs and system outputs of various types, primarily linguistic inputs and outputs such as speech or text, but also potential multimodal inputs and outputs, such as selections on a touchscreen or images presented by a system. Dialog events can be used to express whole utterances, phrases, or other slices of time.

Components of dialog systems may receive an event and add features to it, for example, a natural language interpretation component may receive an event containing a text feature and add a semantic feature to it as an interpretation of the text.

It is anticipated that in the future, events could be joined together to form streams, for example, to represent continuous input or output of a speech-to-text engine.

0.3 A Foundation for Further Specialization

This specification defines a generic format into which dialog features in different formats can be gathered together into a single event. It also describes how such features can reference each other.

It does not define which formats (mime types) should be present, how features should be named, or which features might be required under what circumstances. This will be left to additional specifications, which we term ***derived specifications,*** built upon this one.

For example, dialog events could be used to carry prompt and response information for a dialog system, annotate spoken dialog between two human speakers, or be used as part of an interface to a natural language component in a text processing solution.

We are anticipating that a future OVON standard for representing utterances in a dialog system or dialog history will be developed as one example of a derived specification.

Chapter 1. Specification

1.1 Representation

Dialog event objects will be represented as a JSON [1] object in a string format. The JSON dialog event object is not intended to be a stand-alone document. It is intended to be included in a larger data structure as an object with a re-usable standard structure

1.2 Dialog Event Object

Each dialog event object has a unique ID, is associated with a single speaker (person or machine), spans a period of time, and contains features representing different inter-related aspects of the event.

```
{
    "id": "user-utterance-30",
    "speaker-id": "b5y09lky5KU5",
    "previous-id": "user-utterance-28",
    "span": {
        "start-time": "2022-12-20 15:59:01.246500+00:00",
        "end-offset": "PT0.1045"
    },
    "features": {
        "my-audio-feature": {  … }
        "my-text-feature": {  … }
        "my-semantic-feature": {   … }
        "my-custom-feature": {   … }
    }
}
```

Figure 1. Example of a dialog event container.

Figure 1 shows an example of a dialog event object. Each dialog event object has a unique id and relates to just one speaker identified by the speaker-id. The event can optionally be linked to a previous event object from the same speaker using the previous-id. The span object represents the time-span of the event.

The features: id, speaker-id, span, and features are mandatory. The previous-id object is optional. The features section will typically contain one or many feature objects but may be empty. Each feature object is identified with a key which can have any arbitrary feature name. As dictionary keys, feature names must be unique within a dialog event.

1.3 Span

The span object describes a period of time during which the event occurred. It is this span that distinguishes the dialog event from a visual user-interface event which occurs at a point in time. Span objects must contain either an absolute start-time or a relative start-offset but not both. They can also optionally contain either an end-time or end-offset but not both. Absolute times are represented in ISO 8601 [2] format, more specifically the simpler interoperable variant specified in IETF RFC 3339 [3]. Relative durations are represented in ISO 8601 [2] duration format.

The top-level span object may contain a start-offset rather than an absolute start-time. This start-offset specifies the start time as the time elapsed since an absolute reference time is defined outside of the object. For example, the event might be contained in a dialog history object and this offset might be relative to the start-time of the conversation represented by that object.

As will be explained below, span objects can also be included in individual tokens within features of the dialog event.

OPEN VOICE NETWORK
Voice worthy of user trust

1.4 Feature Objects

```
{
  "id": "user-utterance-30",
  "speaker-id": "b5y09lky5KU5",
  "previous-id": "user-utterance-28",

  "span": {
    "start-time": "2022-12-20 15:59:01.246500+00:00",
    "end-offset": "PT3.1045"
  },

  “features”: {
    "my-audio-feature": {
      "mime-type": "audio/wav",
      "tokens": [
        {
          "value-url": "http://localhost/xyz1234.wav"
        }
      ]
    },

    "my-text-token-feature": {
      "mime-type": "text/plain",
      "lang": "en",
      "encoding": "UTF-8",
      "token-schema": "BertTokenizer.from_pretrained(bert-base-uncased)"
      "tokens": [
        {"value": "what"},
        {"value": "is"},
        {"value": "the"},
        {"value": "weather"},
        {"value": "forecast"},
        {"value": "for"},
        {"value": "tomorrow"}
      ]
    }
  }
}
```

Figure 2. A dialog event containing an audio feature object and a tokenized text feature object.

As a minimum, each feature object must contain a mime-type and an array of token objects named tokens.

Figure 2 shows a simple example of a tokenized text feature. In this example, the mime-type is plain text and the feature is broken into a sequence of tokenized words. Each word is represented by a token object.

This is just one example of how text can be represented in a feature.

The tokens array contains zero or more token objects, and each token object must contain either a value object or a value-url string.

The value-url string references an external document that contains the value. The reference is a string in the format of a Universal Resource Locator (URL) as shown in the my-audio-feature in Figure 2.

The mime-type is mandatory and defines the format of the value object or the document referenced by the value-url. The format of mime-type is defined in IETF RFC-2231 [6]

The object lang defines the language of the feature according to IETF BCP 47 language Tag [4]. This value is optional and remains undefined if not included. This value informs how the feature should be interpreted. If the feature contains multilingual content it is recommended that this value remains undefined or is set to the dominant language of the feature.

The encoding object defines the text encoding within the value objects. Values should be one of "ISO-8859-1" (See ISO/IEC 8859-1 [11]) or "UTF-8" (See RFC3629 [12]). This value is optional and remains undefined if not included.

The token-schema object is included to allow reference to standard sets of token values or symbols for certain features with specific mime-types. For example, it could be used to specify which phonetic pronunciation alphabet is present in a pronunciation feature, or refer to a specific standard set of intents for a given domain in a semantic feature. The format of this object is not mandated and is expected to be defined in derived specifications. This value is optional and remains undefined if not included. Figure 2 shows one example of how the token-schema might be used to reference a specific text tokenizer. Figure 4 shows an example of how token-schema might be used to reference a standard semantic schema. Both examples are illustrative, not normative.

```
    ..
    "my-text-token-feature": {
      "mime-type": "text/plain",
      "tokens": [
        {
          "value": "what",
          "span": {
            "start-offset": "PT0.0210",
            "end-offset": "PT0.1457"
          }
        },
        {
          "value": "is",
          "span": {
            "start-offset": "PT0.1460",
            "end-offset": "PT0.1976"
          }
        },
        ..
        {
          "value": "tomorrow",
          "span": {
            "start-offset": "PT2.3082",
            "end-offset": "PT3.0784"
          }
        },
      ]
    }
    ..
```

Figure 3. A feature object containing tokens objects with spans.

Each token object can also optionally contain a span object. This indicates the time span of this particular token. As described in section 1.3 Span, spans can be defined as absolute times or relative offsets. In a token object start-offset and end-offset are relative to the start-time of the parent dialog event (or the implied dialog event start time defined by its start-offset).

Figure 3 shows an example where the tokenized words of the text feature for Figure 2 are each given an individual span.

1.5 Confidence, Linking, and Stand-Off Annotation

Each feature should represent just one aspect of the dialog event. Each token object within each feature can be linked to another token object in the event using the object links.

```
{
  "id": "user-utterance-30",
  "speaker-id": "b5y09lky5KU5",
  "previous-id": "user-utterance-28",

  "span": {
      "start-time": "2022-12-20 15:59:01.246500+00:00",
      "end-offset": "PT0.1045"
  },

  "features": {
    "my-text-feature": {
      "mime-type": "text/plain",
      "lang": "en",
      "encoding": "UTF-8",
      "tokens": [
        {
          "value": "what is the weather forecast for tomorrow",
          "confidence": 0.99
        }
      ]
    },

    "my-semantic-feature": {
      "mime-type": "text/x.my-semantic-mime-type",
      "token-schema": "standard-intents.org/schemas/weather.xml",
      "tokens": [
        {
          "value": {
            "name": "intent",
            "label": "WeatherForecast"
          },
          "links": ["$.my-text-feature.tokens[0].value"],
          "confidence": 1.0
        },
        {
          "value": {
            "name": "date",
            "label": "02-14-2023"
          },
          "links": [
"$.my-text-feature.tokens[0].value.substring(34,41)" ],
          "confidence": 0.87
        }
      ]
    }
  }
}
```

Figure 4. A simple example of linking between features with confidence assigned to a feature.

Figure 4 shows an example of a semantic feature which links to the text that it represents.

The optional links object is an array of references to the value (or parts of the value) of other token objects using a JSON Path with some extensions. There is no restriction on how many links a feature can have.

Unlike the text in Figure 2, in this example, the text is not tokenized. The semantic token with the name "intent" is linked to the whole text of the utterance as defined by the JSON path $.my-text-feature.tokens[0].value. The semantic token with the name "date" however is linked to a sub-string of the source text - the word ‘tomorrow’ - via the JSON Path $.my-text-feature.tokens[0].value.substring(34,41). Section 1.6 describes how to use JSON Path references in more detail.

The optional confidence object is a number carrying a measure of the confidence that the information contained in the associated value is ‘correct’. Confidence values are expressed as a probability (real number between 0.0 and 1.0). Derivative specifications may override this range for specialized situations.

Recall that the format of a given value object is defined by the mime-type and will vary between features.

Feature layering and cross-referencing are both important parts of the dialog event standard. Together they permit the different features of an utterance or linguistic event to be kept separate but also linked logically and temporally with each other. This approach is termed stand-off-annotation. [9]

1.6 JSON Path and the substring() extension

Individual elements of the linked object are defined as strings containing JSON Paths. JSON Path is a notation for querying and manipulating data stored in JavaScript Object Notation (JSON). At the time of publication, there is not an agreed standard definition for JSON Path. There are a number of implementations based on JSON Path [7] that are broadly compatible. There is also an IETF task force RFC6901 [8] working on a common standard.

JSON Path expressions in links objects are cross-references from one token object to the value, or part of a value, of another token object. For these references, the root document (designated by '$') is the features object in the current dialog event. There is currently no way to specify a link between features that are not contained in the same dialog event.

Implementations should support one additional non-standard extension to JSON Path - namely the substring() function. This extension is added to allow token objects to reference a specific substring within another token object, for example, a word, phrase, or arbitrary sequence of characters.

The substring function can be invoked at the tail end of a path. The input to this function is the output of the preceding path expression. The substring function takes two ordered comma-separated parameters, the index of the start and end character in the substring. The first character of the string is at the index zero.

For example the JSON Path: $.my-text-feature.tokens[0].value.substring(34,41) will return the 34th to the 41st character of string contained within the object referred to by the JSON Path $.my-text-feature.tokens[0].value. If the referred object is not a string, or the substring is out-of-bounds then the resulting behavior is undefined.

Token objects can contain content that is specified externally via a value-url string. If this external content is expressed in JSON format then the JSON path should be interpreted as if the value had been specified locally. This is functionally equivalent to reading the content of the document referenced by value-url and placing it in the equivalent local value object before applying the links reference.

1.7 Alternates

```
{
  "id": "user-utterance-30",
  "speaker-id": "b5y09lky5KU5",
  "span": {
      "start-time": "2022-12-20 15:59:01.246500+00:00"
  },

  "features": {
    "my-text-feature": {
      "mime-type": "text/plain",
      "tokens": [
        {
          "value": "what is the weather forecast for tomorrow",
          "confidence": 0.96
        }
      ],
      "alternates": [
        [
          {
            "value": "what is the weather forecast for thursday",
            "confidence": 0.73
          }
        }
      ]
    }
  }
}
```

Figure 5. Defining alternate values for a feature.

Spoken or written language rarely has a single unambiguous interpretation. For this reason, the dialog event object allows for multiple interpretations of a given event. As has already been seen, the tokens object is used to represent the preferred interpretation of a given feature. In addition to this, the optional alternates array can be used to express alternate interpretations of this feature.   For example, in Figure 5 it can be seen that the preferred interpretation of the feature named my-text-feature has the value *"what is the weather forecast for tomorrow"* and a confidence of 0.96. *The* alternates array contains one additional interpretation of the utterance with the value *"what is the weather forecast for thursday"* and a confidence of 0.73.

The alternates object is an array of an array. Each object in the outer array represents an alternative interpretation and each inner array represents the token objects for this interpretation. These inner token objects have the same format as array elements in the tokens object.

The outer array can contain zero or more interpretations. An empty alternates array implies that there are no alternate interpretations and is equivalent to the alternates object not being present in the event.

1.8 Nomenclature

This specification uses 'kebab-case' (i.e. hyphenated lowercase) for all nominal property names, for example, start-offset and value-url. User-defined object keys such as the dictionary names used in the features section can use any valid JSON expression. It is recommended but not mandated that any normative key names in derived specifications follow the same kebab-case convention.

Chapter 2. Schema

The structure of a JSON dialog event is defined below as a JSON Schema.

```
{
  "$id": "https://openvoicenetwork.org/schema/dialog-event.json",
  "$schema": "https://openvoicenetwork.org/schema",
  "description": "A representation of a 'language event’ that is to say
any information associated with a phrase, utterance or part of an
utterance.",
  "type": "object",
  "required":  ["id", "speaker-id", "span", "features" ],
  "properties": {
    "id": {
      "type": "string"
    },
    "previous-id": {
      "type": "string"
    },
    "speaker-id": {
      "type": "string"
    },
    "span": { "$ref": "#/$defs/span" },
    "features": {
      "type": "object",
      "patternProperties": {
        ".*": { "$ref": "#/$defs/features" }
      }
    }
  },
  "$defs": {
    "features": {
      "type": "object",
      "required":  ["mime-type" , "tokens" ],
      "properties": {
        "encoding": {
          "type": "string",
          "description": "The text encoding of the token values"
        },
        "mime-type": {
          "type": "string",
          "description": "The mime-type of the token values"
        },
        "lang": {
          "type": "string",
          "description": "The language of the token values"
        },
        "token-schema": {
          "type": "object",
          "description": "A schema restricting the token values"
        },
        "tokens" : {
          "type": "array",
          "items":  {
              "$ref": "#/$defs/token"
          },
        "alternates": {
          "type": "array",
          "items": {
            "type": "array",
              "items": {
                "type": "object",
                "$ref": "#/$defs/token"
              }
            }
          }
        }
      }
    },
    "token": {
      "type": "object",
      "anyOf": [
        { "required": ["value"] },
        { "required": ["value-url"] }
      ],
      "properties": {
          "value": {
            "type": ["number","string","object","array","boolean"]
          },
          "value-url": {
            "$ref": "#/$defs/url"
          },
          "confidence": {
            "type": "number"
          },
          "span": { "$ref": "#/$defs/span" },
          "links" : {
              "type": "array",
              "items":  {
                "$ref": "#/$defs/jsonpath"
              }
          }
      }
    },
    "url": {
      "type": "string",
      "description": "Any valid URL"
    },
    "jsonpath": {
      "type": "string",
      "description": "an expression in JSON Path syntax"
    },
    "span": {
      "anyOf": [
        { "required": ["start-time"] },
        { "required": ["start-offset"] }
      ],
      "properties": {
        "start-time": {
            "$ref": "#/$defs/iso-time"
        },
        "end-time": {
            "$ref": "#/$defs/iso-time"
          },
          "start-offset": {
              "$ref": "#/$defs/iso-duration"
          },
          "end-offset": {
              "$ref": "#/$defs/iso-duration"
          }
        }
    },
    "iso-time": {
      "description": "A string in ISO 8601 absolute format."
    },
    "iso-duration": {
      "description": "A string in ISO 8601 duration format."
    }
  }
}
```

Chapter 3. References

[1] https://www.ecma-international.org/publications-and-standards/standards/ecma-404/ ECMA-404 The JSON data interchange syntax
[2] https://www.iso.org/iso-8601-date-and-time-format.html ISO 8601 Date and Time Format.
[3] https://datatracker.ietf.org/doc/html/rfc3339 Newman, Chris; Klyne, Graham (July 2002). Date and Time on the Internet: Timestamps. IETF. doi:10.17487/RFC3339. RFC 3339. Archived
[4] https://www.rfc-editor.org/rfc/rfc5646.txt RFC 5646. BCP 47. Tags for Identifying Languages.
[5] https://www.ietf.org/rfc/rfc4646.txt RFC 4646 Regarding Best Practice for Tags for Identifying Languages
[6] https://datatracker.ietf.org/doc/html/rfc2231 MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations
[7] https://goessner.net/articles/JsonPath/ JSONPath - XPath for JSON. Stefan Goessner.
[8] https://www.rfc-editor.org/rfc/rfc6901 RFC6901. JavaScript Object Notation (JSON) Pointer
[9] Pose-Rodriguez, Javier & Lopez, Patrice & Romany, Laurence. (2014). A Generic Formalism for Encoding Standoff annotations in TEI.
[10] https://openvoicenetwork.org/docs/2832-2/ Interoperable Dialog Packet Requirement Specification.
[11] https://www.iso.org/standard/28245.html ISO/IEC 8859-1:1998 Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1

[12] https://datatracker.ietf.org/doc/html/rfc3629 RFC3629. UTF-8, a transformation format of ISO 10646

Chapter 4. Glossary of Terms

Term	Definition
dialog event	A linguistic event in a spoken or written monologue or dialog between two or more speakers.
dialog event object	A JSON object encoding a dialog event as part of an interface to a natural language component in a text or speech processing solution or dialog system.
dialog event feature	A layer of information of a certain type associated with the dialog event.
confidence	A number representing a measure of the confidence that the information contained in the associated value is ‘correct’.
token encoding	The specific encoding used to represent text in a token.
token link	A link from one token in a feature to part or all of another token in a feature used to implement stand-off annotation.
JSON path	An unambiguous reference to part of a JSON object.
language code	A code representing the language (e.g., American English, British English, New Norsk, etc.)
feature mime type	The type of token values in a specific dialog event feature.
dialog event object id	The unique identifier of the dialog event object
token span	Identifies the span of time for an individual token object
dialog event span	Identifies the span of time for the dialog event

dialog event speaker id	A unique id of the human or machine associated with content of the dialog event
stand-off-annotation	A method of feature layering and cross-referencing that permits the different features of an utterance or linguistic event to be kept separate but also linked logically and temporally with each other.
feature token	A representation of part of the information that makes up a feature value.
token object	A JSON object representing a feature token which defines the value of the feature and other associated information such as its span and how it links to other feature tokens.
derived specification	An derivative standard, built upon this specification, which defines a specific way to use a dialog event in a particular context.

Chapter 5. Decision Log

This section documents some of the key design decisions that were made by the team during the development of this specification. It is informative, not normative.

Issue Topic	Issue and Decision
Nomenclature	Issue: What style of object name should we adopt in the dialog event? Decision: We adopt 'kebab-case' (i.e. hyphenated lower case) for all property names, for example 'start-offset' and 'value-url'.
links	Issue: Should we allow multiple links for a token object? Decision: Yes so that one feature can reference more than one other feature.
Span start and end times	Issue: Should start times and end times be mandatory or optional? Decision: Start times are required. End times should be optional because events containing text to be generated via

	text-to-speech cannot know when the generated text will actually end in real time. For such generative events the start time will be the time that the utterance is intended to start. It cannot anticipate actual delays in processing. We would anticipate that when the text is processed to generate the audio, the audio will have full span. At this point the end time and the start time may be modified to represent the viewpoint of the downstream process that generated it (i.e. the TTS may add an end time and also change the start time to the point where an event is re-transmitted).
Start and end offsets	Issue: Should start and end times just be absolute or should we also support relative options such as offsets? Decision Times can be absolute or relative but not both. The optionality is not complex and supports common use cases.
Time Zones	Issue: Does it matter that ISO can represent different time zones and will it cause a problem if servers are in different time zones? Decision: No. It shouldn't matter as long at consumers of messages regularize for their own time zone before using the time. ISO 8601 carries the time zone information in the format.
Features	Issue: What is the cardinality of features object? Decision: We expect at least one feature but in the spec we will allow for zero or more features.
Speaker Roles	Issue: Should an event carry speaker roles (e.g. agent, customer, bot, etc.)? Decision: No. Because the role of a speaker is defined by the consumer of the event not the event itself
Representation	Issue: Should we be syntax-neutral or mandate a specific representation? (e.g. YML, JSON and XML) Decision: We will specify messages as JSON. It is straightforward to translate it at source and destination into other formats.
Representation of Alternates	Should a feature object contain a simple list of lists of tokens or should we separate out alternates into a different object? Decision: We will treat alternates separately to keep the object

	simple for systems that do not represent uncertainty in their events.
Feature Cross-Referencing	Issue: Should we use simple array indexing or something more sophisticated such as JSON Path? Decision: We will use path indexing via JSON Path. Using a path notation means that we do NOT need to insist that values are arrays. They can be any kind of object. This will allow us to reference strings by character indexes OR reference arrays by array indexes OR reference dictionaries by their dictionary names.
External Links	Issue: How do we refer to features in external events? (This was originally a requirement in the requirements specification.) Decision: We leave this open for now. See comments on streaming.
Feature Span	How do we attach a time span to a feature? Decision: We allow the span object to be added to token objects.

OVON Interoperable Event Restrictions

Confidence	Issue: Should we leave the range of confidence values open for derived specifications to decide? Decision: No. We will mandate a floating point number between 0..1 and leave it up to derived specifications to override this.
Streaming	Issue: Is a stream represented by multiple events in the same stream? Decision: The use of features and events for streaming needs to be worked through more carefully. It is possible that it will require each feature to have a separate ID and span to allow different spans for different features and to allow continuously evolving interpretations in for example an asynchronous dialog system.

	Issue: Does streaming apply to both speech and text? Decision: Yes
JPath or JSON Path?	Should we adopt JPath or JSON Path? Decision: Neither are true standards. JPath appears to be IBM specific. We will adopt JSON Path and extend it to support substring matching. We will also aim to adopt any emerging IETF standard if it is completed. We welcome comments on other standard ways to refer to JSON paths
encoding	Issue: Is it the best approach to follow IETF HTTP in allowing only two encoding types "ISO-8859-1" [ISO-8859-1] or "UTF-8" [RFC3629]. Decision: Yes for now but we welcome comments.

Token array	Issue: This representation
	"tokens":[
	{"token":
	{
	"value": "what",
	"span": {
	---etc
	}
	}
	},
	{"token":
	{
	"value": "what",
	"span": {
	---etc
	}
	}
	}
	]
	is different than the unnamed array elements. This may be valid with JSON indexing. But it permits/forces a JSONtoXML conversion to decide what to "name" the subelements.
	Decision: to be discussed

Value of “tokens”	Issue: In 1.5 Confidence ....
	The value refers to a complete string:
	"tokens": [
	{
	"value": "what is the weather forecast for tomorrow",
	"confidence": 0.99
	}

	] But in the earlier example they were single word "tokens". Will developers need to scan the char-strings to know what they are getting? Decision: to be discussed
Alternates	Issue: . In the "alternates" section Figure 5 is missing "]" and if you wanted it to represent multiple alterates then we should have another alternate array element in the example. This gets verbose if the "tokens" array is single words as in figure 2. Decision: to be discussed
Tokens	It looks like according to the Schema the things under "tokens" and "alternates" must be a "token", but the examples don't reflect that. Decision: to be discussed