ecma INTERNATIONAL

Standard ECMA-430

1st Edition / December 2025

Natural Language Interaction Protocol (NLIP)

Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001

Contents

Page

Introduction

The technology of Artificial Intelligence has the potential to be truly transformative to society. Despite some limitations, the technology is capable of many functions, including but not limited to answering questions, translating, describing and summarizing multi-modal content, generating new content, and summarizing large volumes of information. This enables the creation of intelligent agents that can use AI to analyze data and provide new services.

A much bigger boost to the social benefits of AI technology can be obtained by interactions among different intelligent agents, which may be under the control of different organizations and users. The interaction among intelligent agents can unlock new economic and social value, just like the interactions among various Internet-based services was enabled with the advent of the web browser.

There is a need for a standard common protocol that is used by humans to interact with an intelligent agent, and for intelligent agents to interact with each other. This specification proposes such a protocol.

This document describes the specification of the protocol. The binding of the protocol to underlying base transfer protocols are described in companion documents.

This Ecma Standard was developed by Technical Committee 56 and was adopted by the General Assembly of December 2025.

Natural Language Interaction Protocol (NLIP)

1 Scope

This Standard defines the specifications of Natural Language Interaction Protocol (NLIP), which is an application-level communication protocol defined between AI Agents or between a human and an AI agent. The motivation, the design philosophy, exemplar use-cases, and examples of sample exchanges using the NLIP protocol are out of scope of this Standard.

2 Conformance

A conforming implementation of NLIP must provide and support messages and submessages as described in this specification.

A conforming implementation of NLIP must select a base transfer protocol and conform to the specifications of the binding document of that base transfer protocol listed in Clause 3.

A conforming production deployment of NLIP must select a security profile defined in ECMA-434 and support it.

3 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ECMA-404, *The JSON Data Interchange Syntax*, 2nd edition (December 2017)

ECMA-431, *Binding of the Natural Language Interaction Protocol (NLIP) over HTTP/HTTPS*

ECMA-432, *Binding of the Natural Language Interaction Protocol (NLIP) over WebSocket*

ECMA-433, *Binding of the Natural Language Interaction Protocol (NLIP) over AMQP*

ECMA-434, *Security Profiles for the Natural Language Interaction Protocol (NLIP)*

4 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

4.1
base transfer protocol
a base transfer protocol is a communication protocol between two computer systems which supports an encrypted and authenticated transfer of data across those computer systems. HTTPS is an example of a base transfer protocol.

4.2
JSON message
JSON formatted message is any data that is written to be conformant to ECMA-404.

4.3
data modality
the modality of data refers to the type of information the data is representing. Common examples of data modality include natural language (expressed in human languages), structured text (e.g. JSON, XML, or a computer include language like ECMAScript), video, audio, and binary data.

4.4
multi-modal data
multi-modal data is data that has multiple components, where different components may have different data modalities.

4.5
end-point
an end-point is a computer system or computer software that communicates using NLIP with other computer systems or computer software, on the same or another machine.

4.6
agent
an agent is a software which is capable of processing multimodal information, take actions and generate responses based on such processing using an AI model such as a large neural network. The action and response generation may or may not involve a human in the loop.

4.7
client agent
a client agent is an agent which initiates the transfer of data to another agent.

4.8
server agent
a server agent is an agent which waits for other agents to initiate the transfer of data with it and responds to them.

5 NLIP message formats

This clause describes the format of NLIP messages and the required action of an end-point on the receipt of each type of message or sub-message.

Each message exchanged via NLIP is primarily represented as a JSON message. Each message consists of a first submessage and an optional list of other submessages. Capitalization is not relevant to the name or value of the field.

The binding of the protocol to a base transfer protocol may make special provisions for more efficient transfer of binary and structured content. The specific details for these provisions can be found in the binding documents of NLIP over the base transfer protocol.

In the message and submessage structures, all types (arrays, strings, objects etc.) are represented as defined in ECMA-404 JSON specification.

5.1 First submessage

The first submessage in a NLIP message is a JSON object consisting of the following fields:

5.1.1 MessageType field

The MessageType field is optional. Its value must be a string as defined by ECMA-404. The value of ‘control’ is reserved and indicates a message that is used to negotiate control parameters. Control messages are intended to perform operations related to the non-functional aspect of an end-point, e.g. to query about its applicable

policies, or to negotiate any parameters associated with the end-point configuration or protocol configuration. Other messages are intended to perform operations related to the function of the end-point.

When this field is missing, the server may assume that the message is a data message. If the field has any value other than ‘control’, it should be considered a data message, and its interpretation is left to the end-point receiving the message.

5.1.2 Format field

The format field is required. It is used to specify the format in which the contents field is represented. It is described further in clause 5.3.

5.1.3 Subformat field

The subformat field is required. It is used to specify the subcategory of representation within the format. It is described further in clause 5.3.

5.1.4 Content field

The content field is required. It includes the actual content being sent between the client and the server. It must be represented as specified by the format and subformat field.

5.1.5 Submessages field

This is an optional field. When present, it must be represented as a array defined by ECMA-404 consisting of one or more submessages. Each submessage is described in clause 5.2. Following ECMA-404 specifications, an array is an ordered list of values: the order of the submessages in the submessages field is significant.

5.2 Submessage

The submessage is a JSON object consisting of the following fields.

5.2.1 Label field

The Label field is optional. When present, it must have the key of “Label” without regard of the capitalization of the key. Its value field must be a string as defined by ECMA-404. Its example uses could be to carry its submessage’s sequence number or the “role” information in a LLM chat.

5.2.2 Format field

The format field is required and is a string as defined by ECMA-404. It is used to specify the format in which the contents field is represented. It is described in 5.3.

5.2.3 Subformat field

The subformat field is required and is a string as defined by ECMA-404. It is used to specify the subcategory of representation within the format. It is described for various formats in 5.3.

5.2.4 Content field

This content field is required. It includes the actual content that is being sent between the client and the server. It must be represented as specified by the format and subformat field.

5.3 Format and subformat field values

The format field must have one of the values described in Table 1. The format key is “format” regardless of case. When subformat is shown within angular brackets <…>, the content inside angular is a placeholder for generic value as described in Notes.

Table 1 — List of allowed format fields

JSON Value	Subformat Values	Notes
text	<lang>	The content contains natural language text. The subformat identifies the language that is being used for content.
token	<prefix>_<suffix>	The content is an opaque token interpreted by end-points. The subformat begins with a prefix and may contain _ after the prefix and a suffix. Two prefixes are reserved namely ‘authentication’ and ‘conversation’. These submessages provide a way to annotate a message with different types of tokens. For more details, see clause 6.2.
structured	json, uri, xml, html, <prog>	The content is structured content. It can be json, uri, xml or html. <prog> can be name of any programming language. If the name of a programming language is specified, content field contains code in that programming language. If the programming language is not understood/supported by the end-point receiving it, it should send back a response using format of text informing the peer of the same.
binary	<content>/<encoding>	The subformat field must have content as one of the values of audio, image, sensor, or generic. The encoding must be specific content encoding, e.g. bmp or jpeg for audio, mp3/mov for video etc. A sender may include a . as prefix for encoding. Valid values for subformat include are audio/bmp, audio/.bmp, video/.mp3, generic/.zip etc.
location	text, GPS	The content specifies location. The location may be specified as a text description, or a GPS location.
generic	<ext>	The subformat should include the name of an extension that both client and server support. This allows private clients and servers to define their own private format fields.

All conforming end-points must support the processing of all fields, and support at least the text format field with the subformat of ‘English’.

6 Mandatory exchanges

The following behaviors are required to be supported by an end-point in response to the receipt of various fields.

6.1 Initial request

The initial request must be made by a client agent to an server agent. The client agent must send an NLIP message including the first submessage. The server agent must send a response to the client agent as an NLIP message including the first submessage.

6.2 Submessages with format of token

Submessages can be used to maintain state, authentication information or other usage by the communicating end-points, as needed. Submessages with format field of ‘token’ are required if the end-points are maintaining session state information or authentication information for other end-points in communication.

If any NLIP message containing a submessage that where the format field is ‘token’, the end-point must include the same submessage with the exact same format, subformat and content in its response message to the peer end-point, except when the token was initially created by the end-point.

A NLIP message may contain multiple submessages with format fields of ‘token’. As an example, in a communication, each of the end-points may include their own ‘conversation’ token and ‘authentication’ token that they have created, resulting in two submessages per end-point with the format field of ‘token’ in the message.

A submessage with a field of ‘token’ may also contain a subformat which can be any general string, and not necessarily beginning with ‘conversation’ or ‘authentication’. The use of these tokens is up to the application, and may be used to support group identity in case of multi-party communication.

6.2.1 Token Submessages with subformat beginning with conversation

Either the client agent or the server agent may be the first one to create a submessage with format field of ‘token’ and the subformat field starting with ‘conversation’. The other agent can optionally add another submessage with the format of ‘token’ and the subformat of ‘conversation’ if it wants to maintain its own conversation identifier. The conversation token is intended to maintain the concept of a continuous conversation happening between the different end-point.

An end-point including the token format may choose to add its identity after the ‘_’ character to identify who originated the conversation submessage. The contents of the submessage with format ‘token’ are opaque to the protocol and can be interpreted only by the end-point creating it. The agent may also include its identity after the conversation such as ‘conversation_9.2.3.5’ to indicate that the message was created by the agent with identity 9.2.3.5.

6.2.2 Token Submessages with subformat beginning with authentication

A submessage with format ‘token’ and subformat beginning with ‘authentication’ is intended to exchange authentication credentials with communicating end-points. It may or may not be included in an exchange depending on the authentication requirement of the end-point.

A client agent may send a message to the server agent without a submessage containing the format of ‘token’. A client agent or server agent may create a submessage with the format field of ‘token’ and the subformat field beginning with ‘authentication’.

Any end-point creating the authentication token in the first instance must implement the mechanism to translate the token (when returned by the client) into its appropriate credentials (such as user-name, password, certificates or other mechanisms). The other end-point is obligated to return the authentication token unmodified so that authentication can be performed seamlessly.

6.3 Requests containing control field

If a NLIP end-point receives a message which has the control field set to true, it must respond with a NLIP Message with control field set to true.

6.4 Requests for out of band transfer

Base64 encoding may not always be desirable. If a client has a large binary content that it wants to transmit using a mechanism that is more efficient than JSON Base64 encoding, it must use a base transfer protocol that supports such a mechanism. This request is done by exchanging NLIP request and response pair.

The first message is a request from a client agent to the server agent asking for an end-point with an out of band transfer support. This must be a message with the MessageType field set to ‘control’, and the first submessage should include a text format request for the end-point to send large uploads.

The response to this request message from the server agent should be a message with MessageType field set to ‘control’ containing a submessage with the format field set of ‘structured’ and subformat of a URI. The URI should be to an end-point where the base transfer protocol can be used to transmit the large binary content.

7 Base transfer protocol

A base transfer protocol must satisfy the following properties.

7.1 Encrypted communication

The base transfer protocol must support communication using an encrypted transport. During development of prototypes, the base transfer protocol may be configured to skip encryption. However, any deployed implementation of the NLIP protocol must happen over a communication channel that uses encryption.

7.2 Authentication

The base transfer protocol must implement or enable authentication among the client and server. While the exchange of NLIP messages can happen without authentication between the client and server, the server or the client can ask for authentication. The mechanism for authentication may happen using any existing authentication scheme supported by the base transfer protocol.

7.3 Communication end point

The base transfer protocol must support at least one communication end point for the NLIP server agent to use to receive NLIP messages.

A base transfer protocol may support additional end-points to provide for more efficient communication such as upload of large binary content.

Annex A
(informative)

JSON Schema for NLIP Messages

The schema of NLIP messages is described below:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "NLIP Message Schema",
  "type": "object",
  "properties": {
    "MessageType": {
      "type": "string"
    },
    "Format": {
      "type": "string",
      "enum": ["text", "token", "structured", "binary", "location", "generic"]
    },
    "Subformat": {
      "type": "string"
    },
    "Content": {
      "type": ["string", "number", "object", "array", "boolean", "null"]
    },
    "Submessages": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "Label": { "type": "string" },
          "Format": {
            "type": "string",
            "enum":   ["text",   "token",   "structured",   "binary",   "location",
"generic"]
          },
          "Subformat": { "type": "string" },
          "Content": {
            "type": ["string", "number", "object", "array", "boolean", "null"]
          }
        },
        "required": ["Format", "Subformat", "Content"]
      }
    }
  },
  "required": ["Format", "Subformat", "Content"]
}
```