LionWeb Serialization Format

This document describes the serialization format for LionWeb **version 2024.1** chunks.

Table of Contents

Conventions used in this document . . . 2
Design goals . . . 2
Description . . . 2
Overview of structures . . . 2
- Root structure . . . 3
- Language structure . . . 4
  - Language key . . . 4
  - Language version . . . 4
- Meta-pointer . . . 4
- Node structure . . . 4
  - Id value . . . 5
  - Property . . . 5
  - Containment . . . 5
  - Reference . . . 6
  - Annotation . . . 6
  - Parent . . . 7
- Property serialization . . . 7
  - String . . . 7
  - Boolean . . . 7
  - Integer . . . 7
  - Structured Datatype . . . 8
  - Enumeration literal . . . 11

Examples . . . 11
- Minimal . . . 11
- Minimal node . . . 11
- Property variants . . . 12
- Containment variants . . . 15
- Reference variants . . . 17
- Annotation variants . . . 19

Versions . . . 25
- 2024.1 . . . 25
- 2023.1 . . . 25

JSON Schema for serialization . . . 25

Possible values for `properties`, `containments`, and `references` 30

Conventions used in this document

- *italic* words refer to concepts defined by JSON.
- **bold** words refer to concepts defined in this document.
- `monospaced` words describe verbatim contents of the serialization.
- ALL-CAPS key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP14 (RFC2119, RFC8174) when, and only when, they appear in all capitals, as shown here.
- "processed document" refers to the character sequence that's parsed or written, e.g. a file or network stream.
- Footnotes refer to more discussions and rationale, but are non-normative.

Design goals

We want to provide boring and proven infrastructure, so that innovation can be built on top of it. We do not take any measures to reduce the amount of transmitted data.[1] We strive for shallow structures to enable stream-based processing.

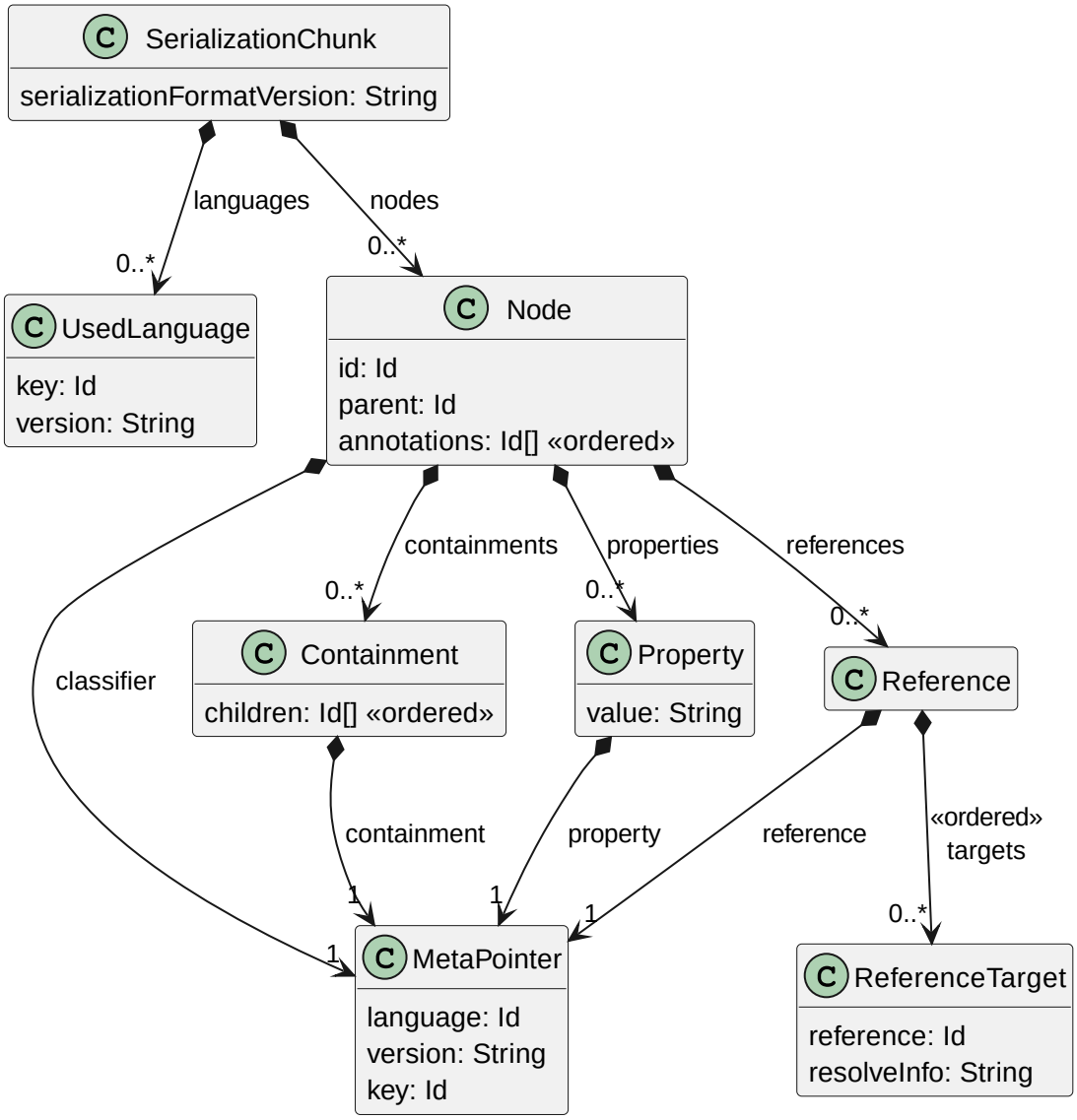
Description

LionWeb node serialization format is defined in JSON (RFC 8259).

We follow the advice of RFC 8259 to be "interoperable", i.e. we assume object keys are unique, and their order is undefined. Any violations SHOULD be reported as error.[2]

Overview of structures

No unspecified *members* are allowed anywhere in the structure.[3]



Root structure

Root level MUST be an *object* with three *members*, called **serialization chunk**.

The first *member* SHOULD be *key* `serializationFormatVersion` with a *string value*.[4][2][5] The value MUST be a non-empty string (without leading or trailing whitespace)[6] describing the serialization format version used to create the processed document, according to Versions.

The second *member* SHOULD be *key* `languages` with an *array value*.[7][8][2] Each *element* in the value array MUST adhere to Language structure. The order of *elements* is undefined. *elements* MUST contain all language/version referred to by any Meta-pointer in the processed document. Each *element* MUST be unique with respect to all its *members*.

The third *member* SHOULD be *key* `nodes` with an *array value*.[9][2] Each *element* in the value array MUST adhere to Node structure. The order of *elements* is undefined. Each *element* MUST be unique with respect to the value of its *key* `id`.

Language structure

Each **used language** MUST be an *object*.[10] The order of *members* is undefined.

NOTE	If the chunk describes a language (M2), it might include instances of builtins' language entities. In this case, builtins MUST be listed as **used language** like any other language.[11]

The *object* MUST contain the following *members*:[7]

- *key* `key` with *string value*, adhering to Language key.
- *key* `version` with *string value*, adhering to Language version.

Language key

A *string* according to Key spec. Refers to the key of the language.

Language version

A *string* with any contents[12][13], MUST NOT be empty.[14] Refers to the version of the language.

Meta-pointer

A **meta-pointer** is a reference from M1 to M2.[15][16] It's used at several places within Node structure.

Each meta-pointer MUST be an *object*. The order of *members* is undefined.

The *object* MUST contain the following *members*:

- *key* `language` with *string value*, adhering to Language key.
- *key* `version` with *string value*, adhering to Language version.
- *key* `key` with *string value* according to Keys spec. Refers to some element in the language. Which element exactly is specified for each usage of meta-pointer.

Node structure

Each **node** MUST be an *object*.[17] The order of *members* is undefined.

The *object* MUST contain the following *members*:[18][19][20]

- *key* `id` with *string value*, adhering to Id value.
- *key* `classifier`[21][22] with *object value*, adhering to Meta-pointer. The **meta-pointer**'s `key`'s *value* refers to the **key** of the **Concept** or **Annotation** this **node** is an instance of.
- *key* `properties` with *array value*, each *element* adhering to Property. The order of *elements* is undefined.[23]
- *key* `containments`[24][25] with *array value*, each *element* adhering to Containment. The order of

elements is undefined.[23]

- *key* `references`[26] with *array value*, each *element* adhering to Reference. The order of *elements* is undefined.[23]
- *key* `annotations`[27] with *array value*, each *element* adhering to Annotation. The order of *elements* MUST be maintained.[28]
- *key* `parent`[29] with *string* or *null value*, adhering to Parent.

For all features (i.e. `properties`, `containments`, and `references`) defined for a **node**'s classifier:[9]

- During serialization, we SHOULD include every feature, even if unset. In the latter case:
  - the **property** MUST have `value` = *null*;
  - the **containment** MUST have `children` = *empty array*;
  - the **reference** MUST have `targets` = *empty array*;
- During deserialization, we MUST accept a **node** even if not all defined features are present.
- We MAY accept undefined features during deserialization if we can deal with them in a meaningful manner.

NOTE	We currently cannot store "invalid text" (i.e. user-entered text that does not adhere to the underlying structure and/or constraints) in the model. We will support this in a future release.[30]

Id value

A *string* according to Identifier spec. Defines the **id** of this **node**.

Property

Each **property** MUST be an *object*. The order of *members* is undefined.

The *object* MUST contain the following *members*:

- *key* `property` with *object value*, adhering to Meta-pointer. The **meta-pointer**'s `key`'s *value* refers to the **key** of the **Property** this **property** is an instance of.
- *key* `value` with *value* as one of
  - *string*[31] containing the value of the property referenced by the `property`. Refer to Property serialization for the specification of the value format. Can be an empty *string*.
  - *null* to explicitly specify the property to be unset.

Containment

Each **containment** MUST be an *object*. The order of *members* is undefined.

The *object* MUST contain the following *members*:

- *key* `containment` with *object value*, adhering to Meta-pointer. The **meta-pointer**'s `key`'s *value*

refers to the **key** of the **Containment** this **containment** is an instance of.

- *key* `children` with *array value* with *string elements*. Each *element* adheres to Identifier spec, and refers to the **id** of the contained **node**. The order of *elements* MUST be maintained.[28]

NOTE	Each `children` element is the inverse relation of a **parent**.

NOTE	The `children` **node** can be contained in the processed document, but also can be outside the processed document (i.e. not contained in the processed document).

Reference

Each **reference** MUST be an *object*. The order of *members* is undefined.

The *object* MUST contain the following *members*:

- *key* `reference` with *object value*, adhering to Meta-pointer. The **meta-pointer**'s `key`'s *value* refers to the **key** of the **Reference** this **reference** is an instance of.
- *key* `targets` with *array value* with *_object elements*. The order of *elements* MUST be maintained.[28] Each *element* MUST have the following *members* in undefined order:[26]
  - *key* `resolveInfo`[32] with *value* as one of:
    - *string* containing **resolveInfo**, a textual hint that might be used to find the target **node** of this reference. Interface INamed SHOULD be used as a default, if available. The exact value depends on the implementation. Can be an empty *string*.
    - *null* if no **resolveInfo** is available.
  - *key* `reference`[33] with *value* as one of:
    - *string* according to Identifier spec. Refers to the **id** of the target **node**.

NOTE	The referred **node** can be contained in the processed document, but also can be outside the processed document (i.e. not contained in the processed document).

    - *null* if the **id** of the target **node** is not known.

Annotation

Each **annotation** MUST be a *string*. It adheres to Identifier spec, and refers to the **id** of the contained annotation **node**.

NOTE	Each **annotation** element is the inverse relation of a **parent**.[27]

NOTE	The annotation **node** can be contained in the processed document, but also can be outside the processed document (i.e. not contained in the processed document).

Parent

One of

- *string* according to Identifier spec. Refers to the **id** of the **node** containing this **node**.

NOTE	**parent** is the inverse relation of either one **containment** or one **annotation**.[34]

NOTE	The referred **node** can be contained in the processed document, but also can be outside the processed document (i.e. not contained in the processed document).

- *null* if
  - This **node** is a **root node**, i.e. this node does not have a parent.
  - This serialization is sent as an update request.

Property serialization

All property values MUST be serialized as JSON *string*.[31][35]. An unset property SHOULD be serialized as JSON *null*.

String

LionCore Strings might be any string, of any length, including (but not limited to):

- empty string: `""`
- only containing whitespace: `" "`
- containing escaped characters as per JSON spec: `"They said:\n \"Hello!\""`
- containing extended Unicode characters: `"𓀐"`
- containing escaped Unicode characters: `"\uD83D\uDE10"`

Boolean

LionCore Booleans MUST be encoded as exactly one of these JSON *strings*:

- `"true"`
- `"false"`

Booleans MUST NOT be encoded with leading or trailing whitespace, uppercase characters, short forms (like `t` or `f`), or decimal representation (like `1`, `0`, `-1`).

Integer

LionCore Integers MUST be encoded as JSON *string*.

- Integers MUST be represented in base-10.
- The digits can be prefixed with either `+` (plus) or `-` (minus).[36]

- Integers MUST NOT be prefixed by leading zeros.
- Integers can contain value zero with any prefix, i.e. `0`, `-0`, or `+0`.
- Integers MUST NOT contain leading or trailing whitespace.
- LionWeb does NOT limit the range of the integer value.[37] An implementation MAY refuse a model containing an integer value outside the supported range.

Examples of valid Integer encodings

- `"0"`
- `"+0"`
- `"-0"`
- `"123"`
- `"-100000"`
- `"+999"`
- `"1000000002000000003000000004000000005000000006000000007000000008000000009000000009999999999"`
- `"-9999999990000000008000000007000000006000000005000000004000000003000000002000000001000000000"`

Examples of invalid Integer encodings

- `""`
- `123`
- `-1`
- `"+-0"`
- `"++1"`
- `"00002"`
- `"0xAA12"`
- `" 5"`
- `"-6 "`

Structured Datatype

LionCore StructuredDataType MUST be encoded as JSON *string*. The string contains a JSON *object* according to spec (RFC 8259) with proper escaping: all double quotes, line breaks, etc. MUST be escaped to form a proper JSON *string*.

The contents of the string are formed as follows: A LionCore StructuredDataType is encoded as JSON *object*. Each Field forms one *member*, with the field's key as JSON *key* and the field's value as JSON *value*. For fields of type String, Boolean, Integer, and Enumeration literal, the *value* is encoded as JSON *string* in the same way as for a property. For fields of type Structured Datatype, the value is encoded as JSON *object*.

Structured datatypes used in the examples

NOTE The format used in this example is non-normative.

```
enumeration Currency                  [id aa, key currency]
  literal EUR                         [id a0, key cur-eur]
  literal GBP                         [id a1, key cur-gbp]

structured datatype Amount            [id 10, key amount]
  value: Integer                      [id 11, key amount-val]
  currency: Currency                  [id 12, key amount-cur]
  digital: Boolean                    [id 13, key digital]

structured datatype Decimal           [id 20, key decimal]
  int: Integer                        [id 21, key decimal-int]
  frac: Integer                       [id 22, key decimal-frac]

structured datatype ComplexNumber     [id 30, key complex]
  real: Decimal                       [id 31, key complex-real]
  imaginary: Decimal                  [id 32, key complex-imaginary]
```

Valid examples

- Amount *42 EUR non-digital*: `"{\n \"amount-val\": \"42\",\n \"amount-cur\": \"cur-eur\",\n \"digital\": \"false\"\n}"`

  unescaped content:

  ```
  {
    "amount-val":  "42",
    "amount-cur": "cur-eur",
    "digital": "false"
  }
  ```

- Decimal *42.0*: `"{\"decimal-int\": \"42\", \"decimal-frac\": \"0\"}"`

  unescaped content:

  ```
  {"decimal-int": "42", "decimal-frac": "0"}
  ```

- ComplexNumber *23.17 + 42.0i*: `"{\n\"complex-real\": { \"decimal-int\": \"23\", \"decimal-frac\": \"17\"},\n\"complex-imaginary\": { \"decimal-int\": \"42\", \"decimal-frac\": \"0\"}\n}"`

  unescaped content:

  ```
  {
  "complex-real": { "decimal-int": "23", "decimal-frac": "17"},
  ```

```
"complex-imaginary": { "decimal-int": "42", "decimal-frac": "0"}
}
```

Invalid examples

- Amount with non-string field values: `"{\n \"amount-val\": 42,\n \"amount-cur\": \"cur-eur\",\n \"digital\": false\n}"`

  unescaped content:

  ```
  {
    "amount-val":  42,
    "amount-cur": "cur-eur",
    "digital": false
  }
  ```

- Decimal with missing field: `"{\"decimal-int\": \"42\"}"`

  unescaped content:

  ```
  {"decimal-int": "42"}
  ```

- Decimal with `null` value: `"{\"decimal-int\": \"42\", \"decimal-frac\": null}"`

  unescaped content:

  ```
  {"decimal-int": "42", "decimal-frac": null}
  ```

- Decimal with missing outer braces: `"\"decimal-int\": \"42\", \"decimal-frac\": \"0\""`

  unescaped content:

  ```
  "decimal-int": "42", "decimal-frac": "0"
  ```

- Decimal with field name as JSON *keys*: `"{\"int\": \"42\", \"frac\": \"0\"}"`

  unescaped content:

  ```
  {"int": "42", "frac": "0"}
  ```

- Decimal with invalid field value: `"{\"decimal-int\": \"42\", \"decimal-frac\": \"nothing\"}"`

  unescaped content:

```
{"decimal-int": "42", "decimal-frac": "nothing"}
```

- Decimal with unknown field: `"{\"decimal-int\": \"42\", \"decimal-frac\": \"0\", \"decimal-comment\": \"life question?\"}"`

  unescaped content:

```
{"decimal-int": "42", "decimal-frac": "0", "decimal-comment": "life question?"}
```

- ComplexNumber with recursively nested structured datatype: `"{\n\"complex-real\": \"{ \\\"decimal-int\\\": \\\"23\\\", \\\"decimal-frac\\\": \\\"17\\\"}\",\n\"complex-imaginary\": { \"decimal-int\": \"42\", \"decimal-frac\": \"0\"}\n}"`

  unescaped content:

```
{
"complex-real": "{ \"decimal-int\": \"23\", \"decimal-frac\": \"17\"}",
"complex-imaginary": { "decimal-int": "42", "decimal-frac": "0"}
}
```

Enumeration literal

LionCore Enumeration literals MUST be encoded as JSON *string value* according to Key spec. MUST refer to the key of an EnumerationLiteral of the Enumeration defined as type of this **Property**.[12]

Examples

Minimal

```
{
  "serializationFormatVersion": "2024.1",
  "languages": [],
  "nodes": []
}
```

Minimal node

```
{
  "serializationFormatVersion": "2024.1",
  "languages": [
    {
      "key": "myLanguage",
      "version": "2"
```

```
    }
  ],
  "nodes": [
    {
      "id": "aaa",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "myConceptId"
      },
      "properties": [],
      "containments": [],
      "references": [],
      "annotations": [],
      "parent": null
    }
  ]
}
```

Property variants

For this example, we need to define an enumeration and a concept that uses the enumeration.

> **NOTE** The format used in this example is non-normative.

Assume this enumeration:

```
enumeration DaysOfWeek [id 23, key days-of-week]

  literal Monday       [id 34, key monday]
  literal Tuesday      [id 2,  key tttt]
  literal Wednesday    [id 55, key 12398712]
```

And this concept:

```
concept OpeningTime           [id 44, key time_to_open]
  property day: DaysOfWeek    [id 42, key day]
  property startHour: Integer [id 22, key starthour]
  property endHour: Integer   [id 89, key endhour]
```

```
{
  "serializationFormatVersion": "2024.1",
  "languages": [
    {
      "key": "myLanguage",
      "version": "2"
```

```
    }
  ],
  "nodes": [
    {
      "id": "bbb",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "myConceptId"
      },
      "properties": [
        {
          "property": {
            "language": "myLanguage",
            "version": "2",
            "key": "stringPropertyId"
          },
          "value": "my string value"
        },
        {
          "property": {
            "language": "myLanguage",
            "version": "2",
            "key": "integerPropertyId"
          },
          "value": "123"
        },
        {
          "property": {
            "language": "myLanguage",
            "version": "2",
            "key": "booleanPropertyId"
          },
          "value": "true"
        },
        {
          "property": {
            "language": "myLanguage",
            "version": "2",
            "key": "structuredDatatypeId"
          },
          "value": "{ \"name\": \"Bob\" }"
        },
        {
          "property": {
            "language": "myLanguage",
            "version": "2",
            "key": "unsetPropertyId"
          },
          "value": null
        }
```

```
      ],
      "containments": [],
      "references": [],
      "annotations": [],
      "parent": null
    },
    {
      "id": "21",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "time_to_open"
      },
      "properties": [
        {
          "property": {
            "language": "myLanguage",
            "version": "2",
            "key": "day"
          },
          "value": "tttt"
        },
        {
          "property": {
            "language": "myLanguage",
            "version": "2",
            "key": "starthour"
          },
          "value": "9"
        },
        {
          "property": {
            "language": "myLanguage",
            "version": "2",
            "key": "endhour"
          },
          "value": "5"
        }
      ],
      "containments": [],
      "references": [],
      "annotations": [],
      "parent": null
    }
  ]
}
```

Containment variants


```
{
  "serializationFormatVersion": "2024.1",
  "languages": [
    {
      "key": "myLanguage",
      "version": "2"
    }
  ],
  "nodes": [
    {
      "id": "ccc",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "myConceptId"
      },
      "properties": [],
      "containments": [
        {
          "containment": {
            "language": "myLanguage",
            "version": "2",
            "key": "emptyContainmentId"
          },
          "children": []
        },
        {
          "containment": {
            "language": "myLanguage",
            "version": "2",
            "key": "singleContainmentId"
          },
          "children": [
            "cdd"
          ]
        },
        {
          "containment": {
            "language": "myLanguage",
            "version": "2",
            "key": "multiContainmentId"
          },
          "children": [
            "cee",
            "cff",
            "cgg"
          ]
        }
```

```
      ],
      "references": [],
      "annotations": [],
      "parent": null
    },
    {
      "id": "cgg",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "differentConceptId"
      },
      "properties": [],
      "containments": [],
      "references": [],
      "annotations": [],
      "parent": null
    },
    {
      "id": "cdd",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "otherConceptId"
      },
      "properties": [],
      "containments": [],
      "references": [],
      "annotations": [],
      "parent": "ccc"
    },
    {
      "id": "cee",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "differentConceptId"
      },
      "properties": [],
      "containments": [],
      "references": [],
      "annotations": [],
      "parent": null
    }
  ]
}
```


node with **id** `cff` is outside the processed document.

Reference variants

We support different kinds of targets.[38]


```
{
  "serializationFormatVersion": "2024.1",
  "languages": [
    {
      "key": "myLanguage",
      "version": "2"
    }
  ],
  "nodes": [
    {
      "id": "ddd",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "myConceptId"
      },
      "properties": [],
      "containments": [],
      "references": [
        {
          "reference": {
            "language": "myLanguage",
            "version": "2",
            "key": "emptyReferenceId"
          },
          "targets": []
        },
        {
          "reference": {
            "language": "myLanguage",
            "version": "2",
            "key": "singleReferenceId"
          },
          "targets": [
            {
              "resolveInfo": "some name",
              "reference": "dee"
            }
          ]
        },
        {
          "reference": {
            "language": "myLanguage",
            "version": "2",
            "key": "multiReferenceId"
          },
```

```
        "targets": [
          {
            "resolveInfo": "self-reference",
            "reference": "ddd"
          },
          {
            "resolveInfo": "only resolve info",
            "reference": null
          }
        ]
      },
      {
        "reference": {
          "language": "myLanguage",
          "version": "2",
          "key": "noResolveInfoReferenceId"
        },
        "targets": [
          {
            "resolveInfo": null,
            "reference": "dee"
          }
        ]
      },
      {
        "reference": {
          "language": "myLanguage",
          "version": "2",
          "key": "neitherResolveInfoNorReferenceId"
        },
        "targets": [
          {
            "resolveInfo": null,
            "reference": null
          }
        ]
      }
    ],
    "annotations": [],
    "parent": null
  },
  {
    "id": "dee",
    "classifier": {
      "language": "myLanguage",
      "version": "2",
      "key": "differentConceptId"
    },
    "properties": [],
    "containments": [],
    "references": [],
```

```
      "annotations": [],
      "parent": null
    }
  ]
}
```

Annotation variants

For this example, we need to define some annotations and their annotated concepts.

NOTE	The format used in this example is non-normative.

Assume these annotations:

```
annotation Docu [id 23, key docuAnn]
  multiple = true
  annotates = IDocumentable
  property docu: String [id 34, key Docu-docu]

annotation ExtendedDocu extends Docu [id 20, key docuExtended]
  property moreDocu: String [id mds, key MDS]

annotation Marker [id 22, key myMarker]
  multiple = false
  annotates = Node

annotation TrashCan [id 99, key throwAway]
  multiple = false
  annotates = Node
  containment trash: 0..* Node  [id 2, key tat]

interface JavaInfo [id 33, key jv]
  property javaVersion: String  [id 33a, key jvA]

annotation MappedToClass implements JavaInfo [id mtc, key MTC]
  multiple = false
  annotates = Classifier
  reference javaClass: 1 BaseLanguageClass [id jjj, key JJJ]


annotation UsesMapping implements JavaInfo [id um, key UM]
  multiple = false
  annotates = Feature
  reference mapping: 1 MappedToClass [id jjj1, key JJJ1]
```

And these concepts:

```
interface IDocumentable [id 50, key 51]
```

```
concept BaseLanguageClass implements IDocumentable [id 60, key 61]

concept OtherLanguageConcept [id 70, key otherLangConc]
  reference usesTrashCan: 0..1 TrashCan [id 72, key usesTrashCan]
```


```
{
  "serializationFormatVersion": "2024.1",
  "languages": [
    {
      "key": "myLanguage",
      "version": "2"
    },
    {
      "key": "BaseLanguage",
      "version": "1"
    },
    {
      "key": "LionWeb-M3",
      "version": "2024.1"
    }
  ],
  "nodes": [
    {
      "id": "ccc",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "61"
      },
      "properties": [],
      "containments": [],
      "references": [],
      "annotations": [
        "marker",
        "docu1",
        "docu2",
        "localTrash"
      ],
      "parent": null
    },
    {
      "id": "marker",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "myMarker"
      },
      "properties": [],
      "containments": [],
```

```
      "references": [],
      "annotations": [],
      "parent": "61"
    },
    {
      "id": "docu1",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "docuAnn"
      },
      "properties": [
        {
          "property": {
            "language": "myLanguage",
            "version": "2",
            "key": "Docu-docu"
          },
          "value": "This is a very important BaseLanguageClass"
        }
      ],
      "containments": [],
      "references": [],
      "annotations": [],
      "parent": "61"
    },
    {
      "id": "docu2",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "docuExtended"
      },
      "properties": [
        {
          "property": {
            "language": "myLanguage",
            "version": "2",
            "key": "Docu-docu"
          },
          "value": "We want to say a few more things about this BaseLanguageClass"
        },
        {
          "property": {
            "language": "myLanguage",
            "version": "2",
            "key": "MDS"
          },
          "value": "Here be dragons"
        }
      ],
```

```
      "containments": [],
      "references": [],
      "annotations": [],
      "parent": "61"
    },
    {
      "id": "localTrash",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "throwAway"
      },
      "properties": [],
      "containments": [
        {
          "containment": {
            "language": "myLanguage",
            "version": "2",
            "key": "tat"
          },
          "children": [
            "old1",
            "old2"
          ]
        }
      ],
      "references": [],
      "annotations": [],
      "parent": "61"
    },
    {
      "id": "old1",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "SomeConcept"
      },
      "properties": [],
      "containments": [],
      "references": [],
      "annotations": [],
      "parent": "localTrash"
    },
    {
      "id": "old2",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "YetAnotherConcept"
      },
      "properties": [],
```

```
      "containments": [],
      "references": [],
      "annotations": [],
      "parent": "localTrash"
    },
    {
      "id": "bbb",
      "classifier": {
        "language": "LionWeb-M3",
        "version": "2024.1",
        "key": "Concept"
      },
      "properties": [],
      "containments": [
        {
          "containment": {
            "language": "LionWeb-M3",
            "version": "2024.1",
            "key": "Classifier-features"
          },
          "children": [
            "bbb-prop"
          ]
        }
      ],
      "references": [],
      "annotations": [
        "javaMapping"
      ],
      "parent": null
    },
    {
      "id": "bbb-prop",
      "classifier": {
        "language": "LionWeb-M3",
        "version": "2024.1",
        "key": "Property"
      },
      "properties": [],
      "containments": [],
      "references": [],
      "annotations": [
        "typeUseMapping"
      ],
      "parent": "bbb"
    },
    {
      "id": "javaMapping",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
```

```
        "key": "MTC"
      },
      "properties": [],
      "containments": [],
      "references": [
        {
          "reference": {
            "language": "myLanguage",
            "version": "2",
            "key": "JJJ"
          },
          "targets": [
            {
              "resolveInfo": null,
              "reference": "javaClass"
            }
          ]
        }
      ],
      "annotations": [],
      "parent": "bbb"
    },
    {
      "id": "typeUseMapping",
      "classifier": {
        "language": "myLanguage",
        "version": "2",
        "key": "UM"
      },
      "properties": [],
      "containments": [],
      "references": [
        {
          "reference": {
            "language": "myLanguage",
            "version": "2",
            "key": "JJJ1"
          },
          "targets": [
            {
              "resolveInfo": null,
              "reference": "javaMapping"
            }
          ]
        }
      ],
      "annotations": [],
      "parent": "bbb-prop"
    },
    {
      "id": "javaClass",
```

```
      "classifier": {
        "language": "BaseLanguage",
        "version": "1",
        "key": "ClassConcept"
      },
      "properties": [],
      "containments": [],
      "references": [],
      "annotations": [],
      "parent": null
    }
  ]
}
```

Versions

NOTE	The term "version" is a bit ambiguous within LionWeb.[5] This section lists all versions as they appear in `serializationFormatVersion` and `Language.version` for languages `LionWeb-M3` and `LionWeb-builtins`.

2024.1

Technical name: `2024.1`

- Repurposed `JSON` primitive datatype serialization for StructuredDataType

Refer to roadmap for details.

2023.1

Technical name: `2023.1`

Initial version. Refer to roadmap for details.

JSON Schema for serialization

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://lionweb.io/serialization.schema.json",
  "title": "LionWeb Serialization",
  "type": "object",
  "properties": {
    "serializationFormatVersion": {
      "type": "string",
      "pattern": "^\\S+(.*\\S)?$"
    },
```

```
    "languages": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "key": {
            "$ref": "#/$defs/key"
          },
          "version": {
            "$ref": "#/$defs/version"
          }
        },
        "required": [
          "key",
          "version"
        ],
        "additionalProperties": false,
        "minProperties": 2,
        "maxProperties": 2
      },
      "uniqueItems": true
    },
    "nodes": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "id": {
            "$ref": "#/$defs/id"
          },
          "classifier": {
            "$ref": "#/$defs/metaPointer"
          },
          "properties": {
            "type": "array",
            "items": {
              "type": "object",
              "properties": {
                "property": {
                  "$ref": "#/$defs/metaPointer"
                },
                "value": {
                  "oneOf": [
                    {
                      "type": "string"
                    },
                    {
                      "type": "null"
                    }
                  ]
                }
```

```
        },
        "required": [
          "property",
          "value"
        ],
        "additionalProperties": false,
        "minProperties": 2,
        "maxProperties": 2
      }
    },
    "containments": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "containment": {
            "$ref": "#/$defs/metaPointer"
          },
          "children": {
            "type": "array",
            "items": {
              "$ref": "#/$defs/id"
            },
            "uniqueItems": true
          }
        },
        "required": [
          "containment",
          "children"
        ],
        "additionalProperties": false,
        "minProperties": 2,
        "maxProperties": 2
      }
    },
    "references": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "reference": {
            "$ref": "#/$defs/metaPointer"
          },
          "targets": {
            "type": "array",
            "items": {
              "type": "object",
              "properties": {
                "resolveInfo": {
                  "oneOf": [
                    {
```

```
                        "type": "string"
                      },
                      {
                        "type": "null"
                      }
                    ]
                  },
                  "reference": {
                    "oneOf": [
                      {
                        "$ref": "#/$defs/id"
                      },
                      {
                        "type": "null"
                      }
                    ]
                  }
                },
                "required": [
                  "resolveInfo",
                  "reference"
                ],
                "additionalProperties": false,
                "minProperties": 2,
                "maxProperties": 2
              }
            }
          },
          "required": [
            "reference",
            "targets"
          ],
          "additionalProperties": false,
          "minProperties": 2,
          "maxProperties": 2
        }
      },
      "annotations": {
        "type": "array",
        "items": {
          "$ref": "#/$defs/id"
        },
        "uniqueItems": true
      },
      "parent": {
        "oneOf": [
          {
            "$ref": "#/$defs/id"
          },
          {
            "type": "null"
```

```
            }
          ]
        }
      },
      "required": [
        "id",
        "classifier",
        "properties",
        "containments",
        "references",
        "annotations",
        "parent"
      ],
      "additionalProperties": false,
      "minProperties": 7,
      "maxProperties": 7
    },
    "uniqueItems": true
  }
},
"required": [
  "serializationFormatVersion",
  "languages",
  "nodes"
],
"additionalProperties": false,
"minProperties": 3,
"maxProperties": 3,
"$defs": {
  "id": {
    "type": "string",
    "minLength": 1,
    "pattern": "^[a-zA-Z0-9_-]+$"
  },
  "key": {
    "$ref": "#/$defs/id"
  },
  "version": {
    "type": "string",
    "minLength": 1
  },
  "metaPointer": {
    "type": "object",
    "properties": {
      "language": {
        "$ref": "#/$defs/key"
      },
      "version": {
        "$ref": "#/$defs/version"
      },
      "key": {
```

```
          "$ref": "#/$defs/key"
        }
      },
      "required": [
        "language",
        "version",
        "key"
      ],
      "additionalProperties": false,
      "minProperties": 3,
      "maxProperties": 3
    }
  }
}
```

Possible values for `properties`, `containments`, and `references`

Only bold entries are valid.[19]

1 A **Contents**	B `properties: {_ _ _}`	C `containments: {_ _ _}`	D `references: {_ _ _}`
2 `"a": "b"`	**property with id `a` has value `b`**	`containments` value must be array	`references` value must be array
3 `"c": ""`	**property with id `c` has value (empty string)**		
4 `"d": " "`	**property with id `d` has value `` ` `` (one space)**		
5 `"e": null`	**property with id `e` has no value**		
6 (key `f` not present)	**property with id `f` has no value**	**containment with id `f` does not contain any nodes**	**reference with id `f` does not point to any nodes**

<table>
<tr><th>1 A
Contents</th><th>B
properties: {_ _ _}</th><th>C
containments: {_ _ _}</th><th>D
references: {_ _ _}</th></tr>
<tr><td>7 "g": []</td><td rowspan="10">properties value must be string</td><td>containment with id g does not contain any nodes</td><td>reference with id g does not point to any nodes</td></tr>
<tr><td>8 "h": ["i"]</td><td>containment with id h contains node with id `i`</td><td>references value array element must be object</td></tr>
<tr><td>9
<pre>"j": [
{
"resolveInfo": "k",
"reference": "l"
}
]</pre></td><td rowspan="2">containments value array element must be string</td><td>reference with id j points to node with id l, re-binding supported by text k</td></tr>
<tr><td>10 "m": [null]</td><td>references value array element must be an object</td></tr>
<tr><td>11 "n": true</td><td rowspan="5">containments value must be array</td><td rowspan="5">references value must be array</td></tr>
<tr><td>12 "o": 12</td></tr>
<tr><td>13 "p": 34.56</td></tr>
<tr><td>14 "q": {}</td></tr>
<tr><td>15 "r": {...}</td></tr>
<tr><td>16 "s": foo</td><td colspan="3" rowspan="2">JSON syntax error</td></tr>
<tr><td>17 "t": undefined</td></tr>
</table>

[1] We don't care about serialization verbosity #73
[2] Don't rely on valid, but ambiguous JSON structures #159
[3] Allow additional info in serialization #67
[4] Include serialization format version in serialization #58
[5] Use term `release` instead of `version` for LionWeb? #172
[6] Version scheme for LionWeb? #165
[7] Should serialization contain a list of used languages? #76
[8] Rename M3 Metamodel to Language? #78
[9] Repo API: Node representation #33
[10] Establish name for entries in serialization/languages? #129
[11] Details on builtin language #153
[12] Refer to EnumLiteral by key? #128
[13] What does Language.version mean semantically? #130
[14] Add version property to M3 Metamodel #92
[15] Establish term meta-pointer #89

[16] Meta-Object Facility, also known as M3 model

[17] Repo API: Node serialization #37

[18] Require empty members in serialization #59

[19] Require empty members in serialization #33

[20] Always provide both containment and parent id in serialization #55

[21] Discussion on name `concept`

[22] Rename `concept` to `classifier` in node serialization #184

[23] Keep serialization order between different features? #156

[24] Discussion on name `children`

[25] rename `nodes.children` → `nodes.containments`? #206

[26] Discussion on names `references` and `reference`

[27] How to represent annotations in serialization #150

[28] Maintain order in serialization of each containment / reference / annotation #157

[29] Name of `parent` field in serialization #187

[30] How to store invalid text typed at arbitrary places? #62

[31] Repo API: Property value encondings #34

[32] Repo API: Store additional resolve info? #36

[33] Repo API: Represent dangling pointers #35

[34] In serialization, `parent` is inverse of either `children` or `annotations` #186

[35] Supported built-in primitive types #9

[36] Do we allow + prefix for integer property values? #100

[37] Don’t specify minimum supported range for integer #149

[38] Supported reference targets #57