NICE, October 2019

Panel on Systems

Theme
Systems Integration: Bumps and Hopes

Panelists

Moderator

Stephen Clyde, Utah State University, USA

Panelists

Ian Schlarman, Candea LLC, USA
Ivan Krejci, College of Polytechnics Jihlava, Czech Republic
Fabien Mieyeville, Ampere Laboratory, UMR CNRS 5005, France
Mihaela Iridon, Candea LLC, USA

Multiturn Absolute Angular Position Sensor

Ivan Krejčí

College of Polytechnics Jihlava

What is the absolute position sensor?

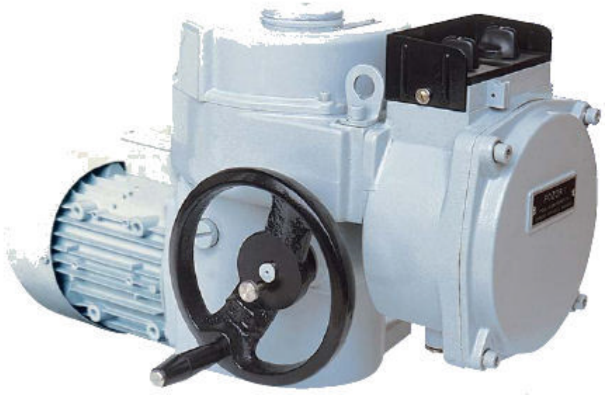
- The sensor must remember its last position when the equipment is switched off.

- If the position is changed when the equipment is switched off (manual handling), the sensor must show the new position after the equipment is switched on.

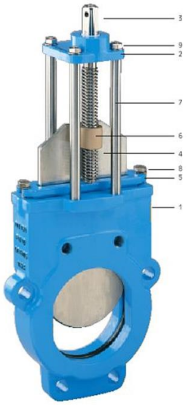
- The sensor must contain a memory element. The memory can be the mechanical one (the gear box) or the semiconductor one (reserve battery needed).

Where are these sensors required?

- In actuating mechanisms for the fluids control



- These actuators control the valve position. Some of these valves need many turns of the driving shaft:



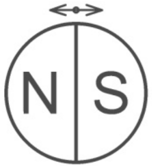
The possible solutions

- The mechanical memory – the gear box requires one turn absolute angular position sensor on each axis of the transmission gearing. Single-turn sensors take advantage of the optical or magnetic (Hall-effect) principles. The actual position can be calculated from all sensors data.

  Advantage: it does not require any back-up battery.

  Disadvantages: complicate construction, complicate position calculation, gear box errors, price.

  The magnet for the single-turn sensor:

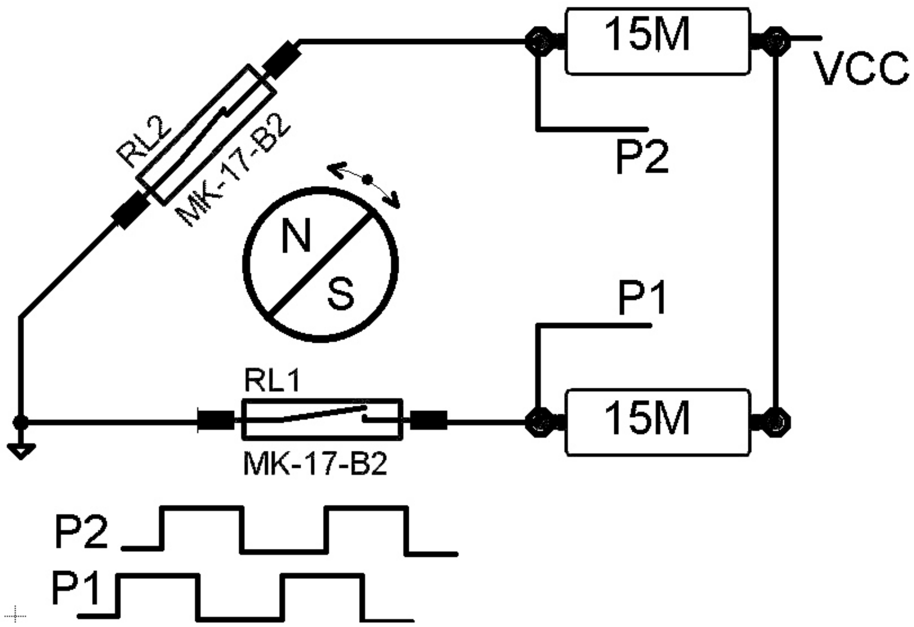


- The electrical solution uses a semiconductor memory. The system takes advantage of one magnetic single-turn absolute sensor and one magnetic field controlled two-bit encoder. The encoder is created by a pair of reed contacts that make an angle 45°. If the magnet turns, switches the reed contacts. Number of switchings is stored in the built-in microprocessor memory. Number of switchings and the single-turn sensor data determine the sensor position.

  Advantage: simple construction, the only mechanical element – the magnet keeper, low price.

  Disadvantage: Back-up battery and low power electronic required.

Reed contact encoder



P1 – Basic contact P2 – Direction contact

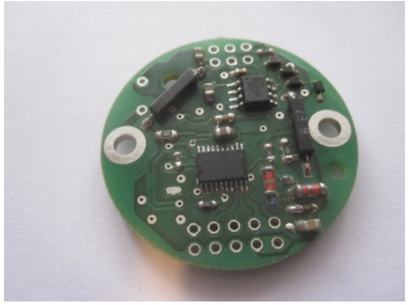
The sensor realization

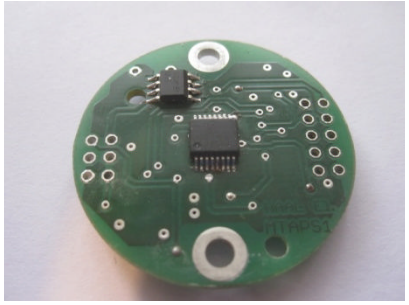
The electrical solution was selected.

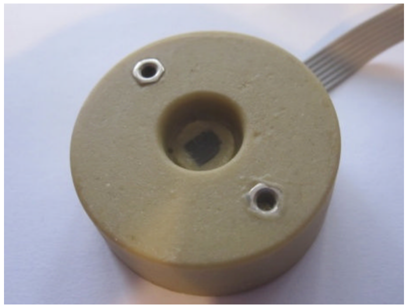
The parameters achieved:

Main components: MCU MSP430F1122, single turn 12b sensor AS5045, reed contacts MK-17-B2

Number of turns	16 k
Resolution	0.07 °
Back-up battery voltage	3 V
Power consumption in sleeping mode	<1 μA
Battery life (supposed) in the back-up mode	> 5 years
Interface:	SPI or UART use of RS485 levels







Building Better Integration APIs

MIHAELA IRIDON
IAN SCHLARMAN

CÂNDEA LLC (TEXAS, USA)

Discussion Points

- ✓ Exposing Data and Behavior
  - ○ General Goals for effective & efficient integration
- ✓ Consumption-Friendly APIs
  - ○ Qualities that make APIs/SDKs easy to understand, consume, test
- ✓ REST APIs Modeling
  - ○ Semantics and Structure Consideration
- ✓ Brief Comparative Study (REST APIs)
  - ○ Merriam Webster vs Oxford English Dictionary APIs
  - ○ Structure of Resource Models; Documentation: generated vs. curated

API Architecture:
Design Drivers & Goals

- ✓ API: Abstraction over some Domain, exposing
  - Data
  - Behavior
- ✓ Target Consumption
  - Open/public
  - Internal
- ✓ Access mechanism/channel
  - REST: Web/HTTP(S)
  - SOAP: Sockets, HTTP, …
  - Messaging

- ✓ Goals (Developer Experience)
  - Reusability
  - Consistency
  - Stability
  - Smooth evolution (versioning)
  - Testability, discoverability
  - Understanding of the underlying Domain (documentation, unambiguous semantics)
  - Ease of troubleshooting (error messages)
    - Visibility (logging)

Web APIs & SDKs

STRUCTURE

Resource Models: Structural Considerations

- ✓ Composition hierarchies
  - FLAT vs HIERARCHICAL
- ✓ Validations (POST & PUT)
  - Custom Frameworks; Rule-based validation rules: how to externalize validation rules (configurability)
  - Meaningful error messages: validate all input vs. stop at first invalid field
- ✓ Access to similar data (REST)
  - Custom routes & inheritance

- ✓ Redundancy for the sake of clarity/model semantics
  - E.g., the use of enumerations in REST models:
    - Use integer values (devoid of semantics), or string values (clarity/self-documenting data), or both?
- ✓ Inheritance in API Controllers
  - Custom routes?
  - Disambiguation?
- ✓ Inheritance in Resource Models
  - Custom deserialization?
  - Disambiguation?

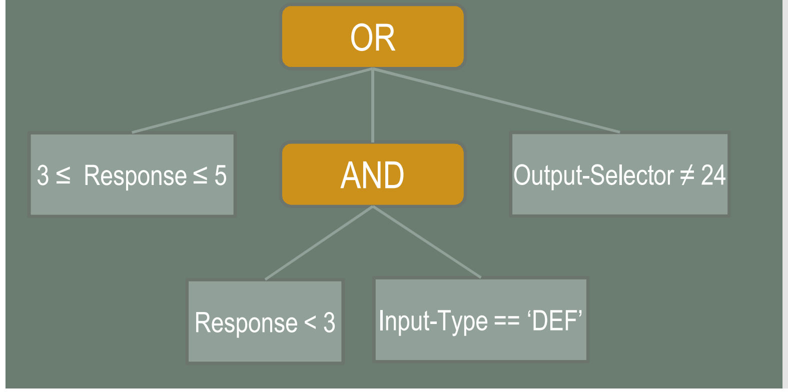
Hierarchical versus Flat Models

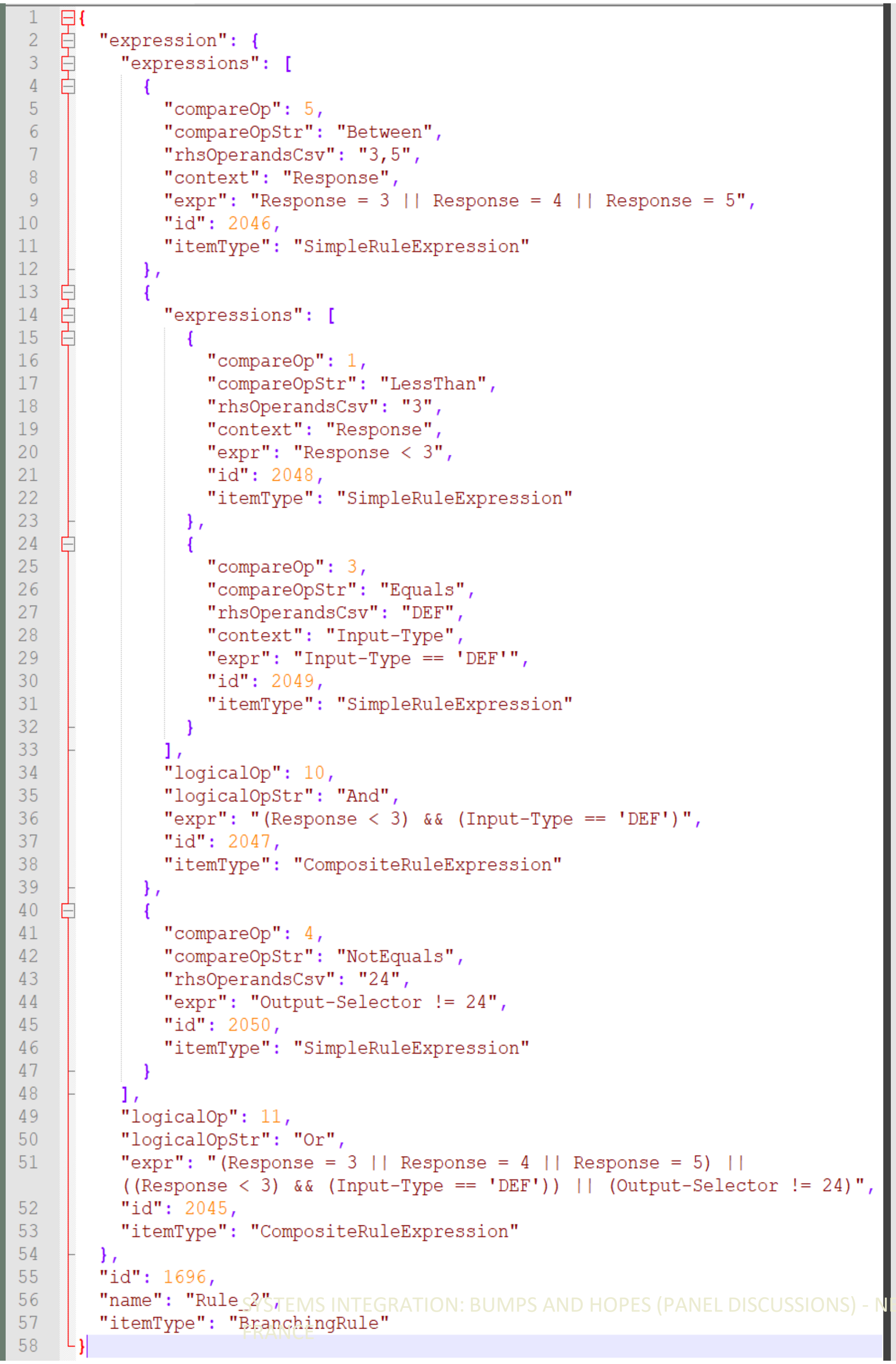
Flat: better suited for REST

Hierarchical: better suited for SDKs, direct access libraries; Domain models.

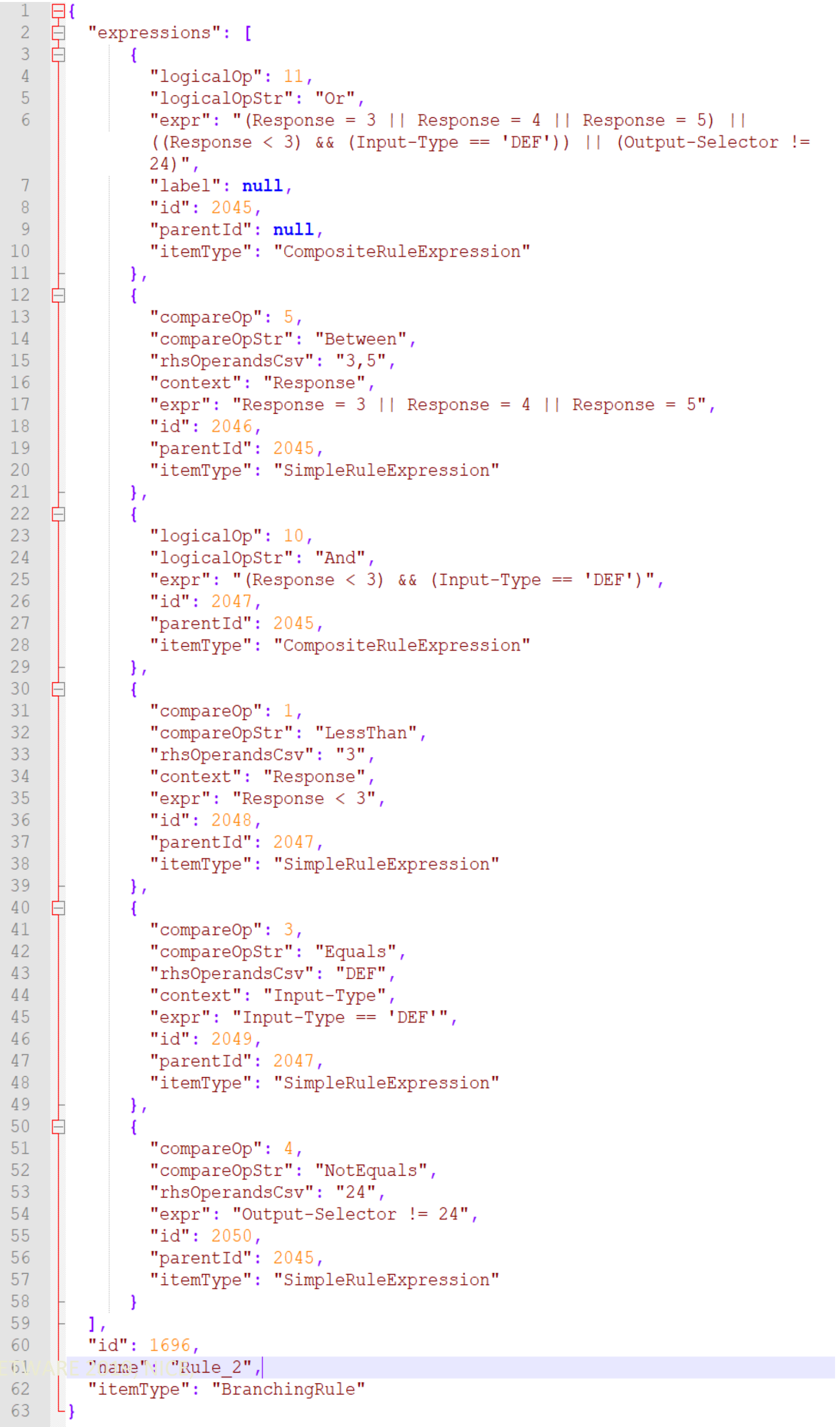
EXAMPLE:

(Response = 3 || Response = 4 || Response = 5) || ((Response < 3) && (Input-Type == 'DEF')) || (Output-Selector != 24)





```
{
  "expression": {
    "expressions": [
      {
        "compareOp": 5,
        "compareOpStr": "Between",
        "rhsOperandsCsv": "3,5",
        "context": "Response",
        "expr": "Response = 3 || Response = 4 || Response = 5",
        "id": 2046,
        "itemType": "SimpleRuleExpression"
      },
      {
        "expressions": [
          {
            "compareOp": 1,
            "compareOpStr": "LessThan",
            "rhsOperandsCsv": "3",
            "context": "Response",
            "expr": "Response < 3",
            "id": 2048,
            "itemType": "SimpleRuleExpression"
          },
          {
            "compareOp": 3,
            "compareOpStr": "Equals",
            "rhsOperandsCsv": "DEF",
            "context": "Input-Type",
            "expr": "Input-Type == 'DEF'",
            "id": 2049,
            "itemType": "SimpleRuleExpression"
          }
        ],
        "logicalOp": 10,
        "logicalOpStr": "And",
        "expr": "(Response < 3) && (Input-Type == 'DEF')",
        "id": 2047,
        "itemType": "CompositeRuleExpression"
      },
      {
        "compareOp": 4,
        "compareOpStr": "NotEquals",
        "rhsOperandsCsv": "24",
        "expr": "Output-Selector != 24",
        "id": 2050,
        "itemType": "SimpleRuleExpression"
      }
    ],
    "logicalOp": 11,
    "logicalOpStr": "Or",
    "expr": "(Response = 3 || Response = 4 || Response = 5) || ((Response < 3) && (Input-Type == 'DEF')) || (Output-Selector != 24)",
    "id": 2045,
    "itemType": "CompositeRuleExpression"
  },
  "id": 1696,
  "name": "Rule_2",
  "itemType": "BranchingRule"
}
```



```
{
  "expressions": [
    {
      "logicalOp": 11,
      "logicalOpStr": "Or",
      "expr": "(Response = 3 || Response = 4 || Response = 5) || ((Response < 3) && (Input-Type == 'DEF')) || (Output-Selector != 24)",
      "label": null,
      "id": 2045,
      "parentId": null,
      "itemType": "CompositeRuleExpression"
    },
    {
      "compareOp": 5,
      "compareOpStr": "Between",
      "rhsOperandsCsv": "3,5",
      "context": "Response",
      "expr": "Response = 3 || Response = 4 || Response = 5",
      "id": 2046,
      "parentId": 2045,
      "itemType": "SimpleRuleExpression"
    },
    {
      "logicalOp": 10,
      "logicalOpStr": "And",
      "expr": "(Response < 3) && (Input-Type == 'DEF')",
      "id": 2047,
      "parentId": 2045,
      "itemType": "CompositeRuleExpression"
    },
    {
      "compareOp": 1,
      "compareOpStr": "LessThan",
      "rhsOperandsCsv": "3",
      "context": "Response",
      "expr": "Response < 3",
      "id": 2048,
      "parentId": 2047,
      "itemType": "SimpleRuleExpression"
    },
    {
      "compareOp": 3,
      "compareOpStr": "Equals",
      "rhsOperandsCsv": "DEF",
      "context": "Input-Type",
      "expr": "Input-Type == 'DEF'",
      "id": 2049,
      "parentId": 2047,
      "itemType": "SimpleRuleExpression"
    },
    {
      "compareOp": 4,
      "compareOpStr": "NotEquals",
      "rhsOperandsCsv": "24",
      "expr": "Output-Selector != 24",
      "id": 2050,
      "parentId": 2045,
      "itemType": "SimpleRuleExpression"
    }
  ],
  "id": 1696,
  "name": "Rule_2",
  "itemType": "BranchingRule"
}
```

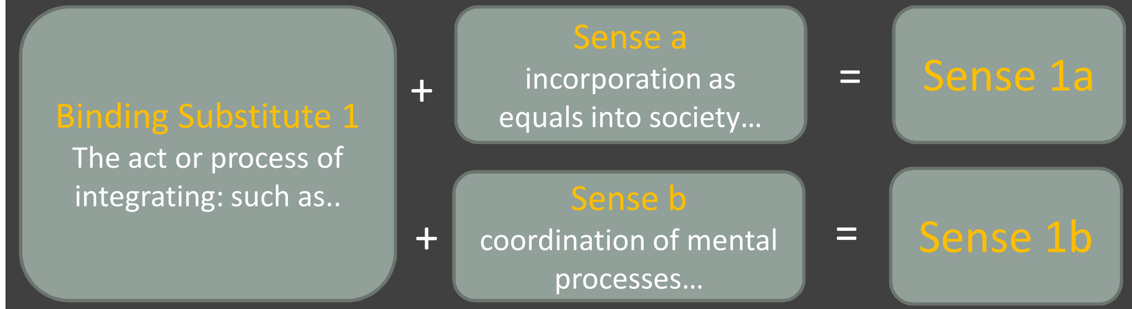
REST: Resource Models

MERRIAM WEBSTER API V. OXFORD DICTIONARIES API

Merriam-Webster Dictionary API

Definition section of a *Headword* Resource

- Deep Hierarchies
- Emphasis on Information Density
  - Abbreviated Property Names
  - Partial Models
- Complex Custom Deserialization
  - Loosely-Typed Models
  - [“type”, object] pattern
  - Object graph traversal to restore semantics



Binding Substitute 1
The act or process of integrating: such as..

+

Sense a
incorporation as equals into society…

=

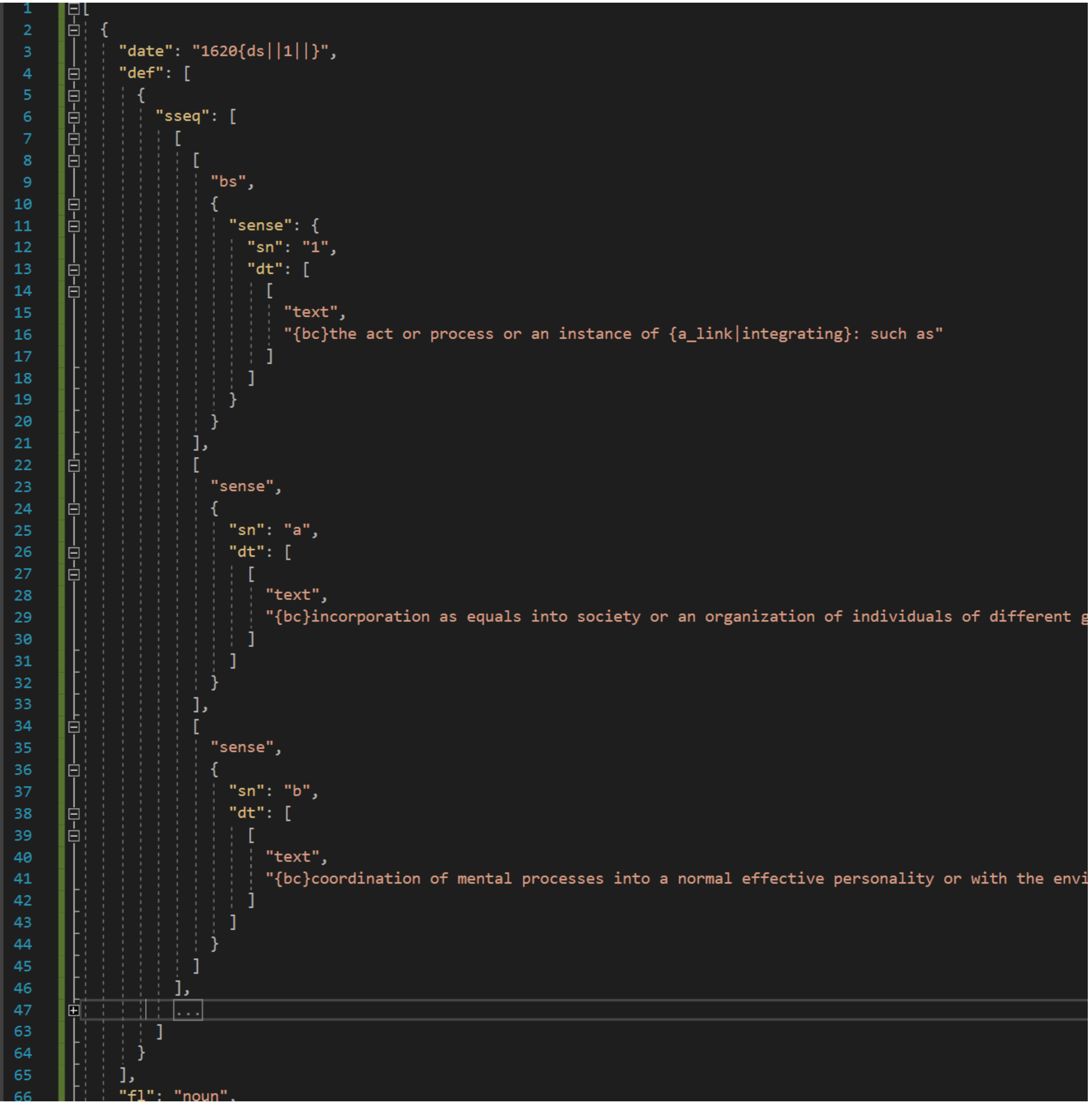
Sense 1a

+

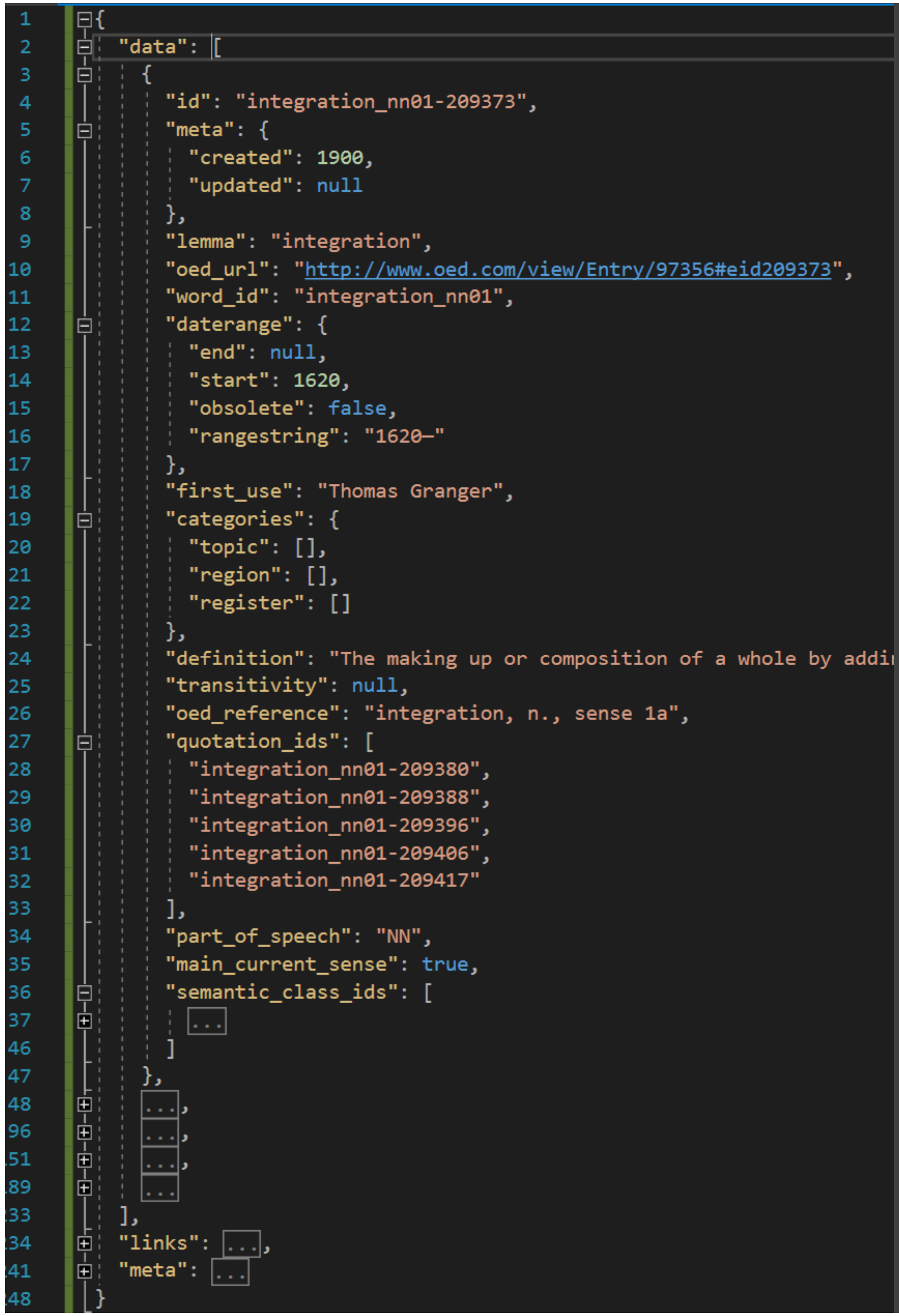
Sense b
coordination of mental processes…

=

Sense 1b



```
[
  {
    "date": "1620{ds||1||}",
    "def": [
      {
        "sseq": [
          [
            [
              "bs",
              {
                "sense": {
                  "sn": "1",
                  "dt": [
                    [
                      "text",
                      "{bc}the act or process or an instance of {a_link|integrating}: such as"
                    ]
                  ]
                }
              }
            ],
            [
              "sense",
              {
                "sn": "a",
                "dt": [
                  [
                    "text",
                    "{bc}incorporation as equals into society or an organization of individuals of different g
                  ]
                ]
              }
            ],
            [
              "sense",
              {
                "sn": "b",
                "dt": [
                  [
                    "text",
                    "{bc}coordination of mental processes into a normal effective personality or with the envi
                  ]
                ]
              }
            ]
          ],
          ...
        ]
      }
    ],
    "fl": "noun",
```



```
{
  "data": [
    {
      "id": "integration_nn01-209373",
      "meta": {
        "created": 1900,
        "updated": null
      },
      "lemma": "integration",
      "oed_url": "http://www.oed.com/view/Entry/97356#eid209373",
      "word_id": "integration_nn01",
      "daterange": {
        "end": null,
        "start": 1620,
        "obsolete": false,
        "rangestring": "1620–"
      },
      "first_use": "Thomas Granger",
      "categories": {
        "topic": [],
        "region": [],
        "register": []
      },
      "definition": "The making up or composition of a whole by addi
      "transitivity": null,
      "oed_reference": "integration, n., sense 1a",
      "quotation_ids": [
        "integration_nn01-209380",
        "integration_nn01-209388",
        "integration_nn01-209396",
        "integration_nn01-209406",
        "integration_nn01-209417"
      ],
      "part_of_speech": "NN",
      "main_current_sense": true,
      "semantic_class_ids": [
        ...
      ]
    },
    ...,
    ...,
    ...,
    ...
  ],
  "links": ...,
  "meta": ...
}
```

Oxford Dictionaries API

Senses custom route on a *Word* Resource

- Strongly-Typed
  - Proxy Models can be easily generated
  - No custom deserialization
- Flattened Hierarchy
  - Incorporates content from parent (Word) Resource
  - Standalone at the expense of verbosity
  - Object Graph hierarchy can be restored without additional content
- Descriptive Naming
- Resource Segregation
  - Words/{word_id}/Senses
  - Preserves relational semantics

Web APIs and SDKs

DOCUMENTATION

Endpoint/Resource Documentation

Generated versus Curated

GENERATED

Consumption

- ✓ Produces Standardized Artifacts
- ✓ Simplifies Content Duplication (not Code Duplication)
- ✓ Removes “Human Error”

Implementation

- ✓ Is Self-Updating (via code introspection)
- ✓ Adds Time for Setup/Customization
- ✓ Introduces Metadata Clutter
- ✓ Adds a Dependency on an External Framework

CURATED

Consumption

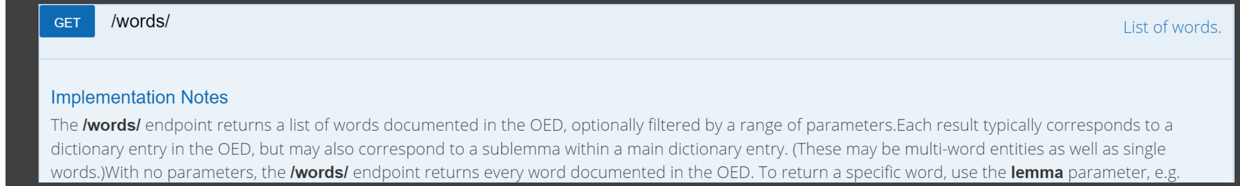
- ✓ Better Conveys Semantics
- ✓ Allows Adding Examples to Highlight “Special” Cases
- ✓ Is Prone to Error

Implementation

- ✓ Requires Manual Updates
- ✓ Enables Contract-First Implementation

Oxford Dictionaries

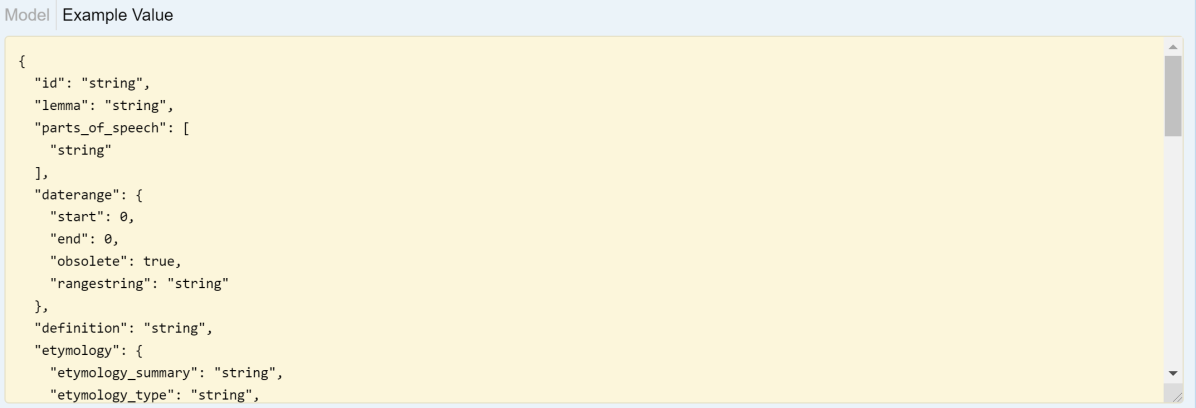
Swagger API Documentation



GET /words/ List of words.

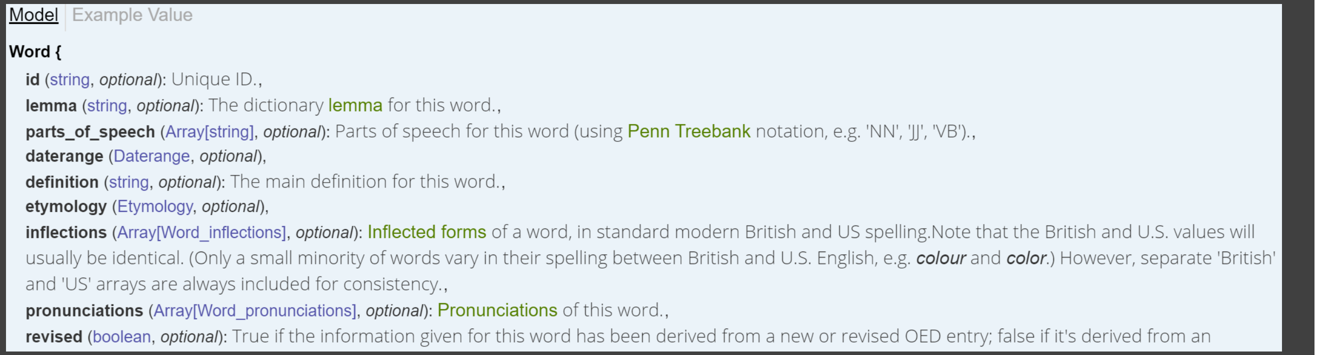
Implementation Notes

The **/words/** endpoint returns a list of words documented in the OED, optionally filtered by a range of parameters.Each result typically corresponds to a dictionary entry in the OED, but may also correspond to a sublemma within a main dictionary entry. (These may be multi-word entities as well as single words.)With no parameters, the **/words/** endpoint returns every word documented in the OED. To return a specific word, use the **lemma** parameter, e.g.



Model Example Value

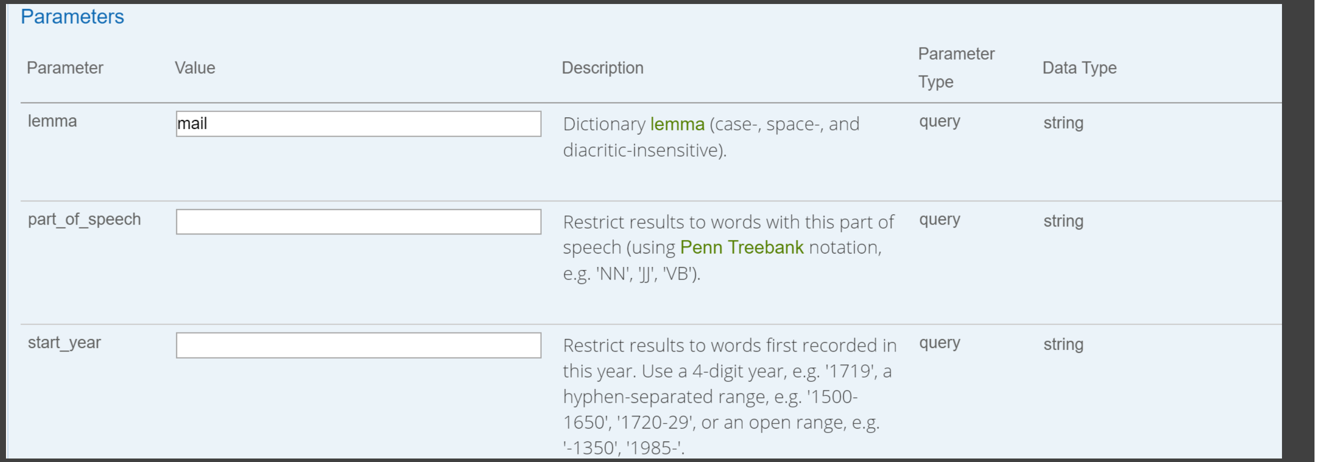
```
{
  "id": "string",
  "lemma": "string",
  "parts_of_speech": [
    "string"
  ],
  "daterange": {
    "start": 0,
    "end": 0,
    "obsolete": true,
    "rangestring": "string"
  },
  "definition": "string",
  "etymology": {
    "etymology_summary": "string",
    "etymology_type": "string",
```



Model Example Value

Word {

id (string, *optional*): Unique ID.,
lemma (string, *optional*): The dictionary lemma for this word.,
parts_of_speech (Array[string], *optional*): Parts of speech for this word (using Penn Treebank notation, e.g. 'NN', 'JJ', 'VB').,
daterange (Daterange, *optional*),
definition (string, *optional*): The main definition for this word.,
etymology (Etymology, *optional*),
inflections (Array[Word_inflections], *optional*): Inflected forms of a word, in standard modern British and US spelling.Note that the British and U.S. values will usually be identical. (Only a small minority of words vary in their spelling between British and U.S. English, e.g. ***colour*** and ***color***.) However, separate 'British' and 'US' arrays are always included for consistency.,
pronunciations (Array[Word_pronunciations], *optional*): Pronunciations of this word.,
revised (boolean, *optional*): True if the information given for this word has been derived from a new or revised OED entry; false if it's derived from an



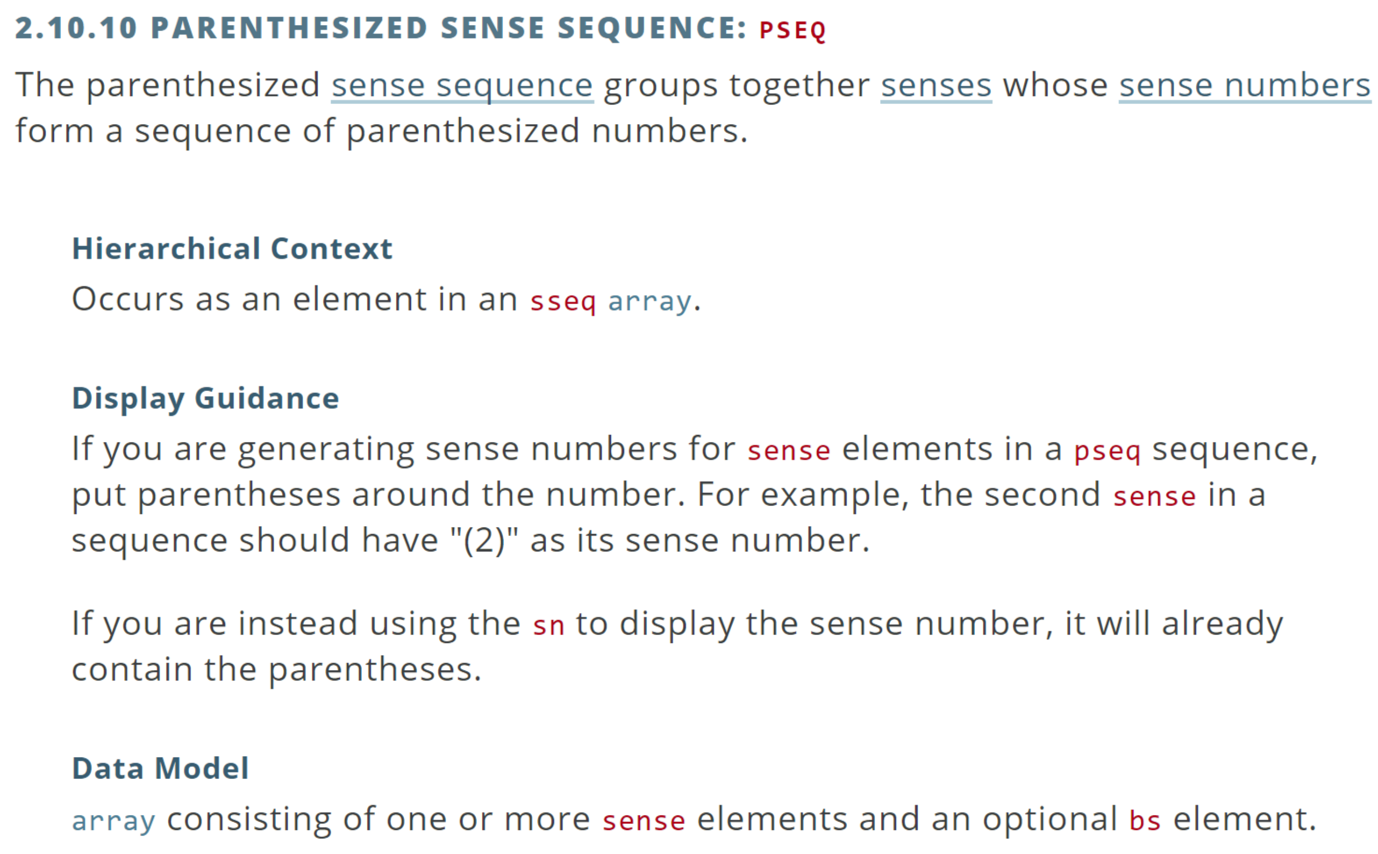
Parameters

Parameter	Value	Description	Parameter Type	Data Type
lemma	mail	Dictionary lemma (case-, space-, and diacritic-insensitive).	query	string
part_of_speech		Restrict results to words with this part of speech (using Penn Treebank notation, e.g. 'NN', 'JJ', 'VB').	query	string
start_year		Restrict results to words first recorded in this year. Use a 4-digit year, e.g. '1719', a hyphen-separated range, e.g. '1500-1650', '1720-29', or an open range, e.g. '-1350', '1985-'.	query	string

https://developer.oxforddictionaries.com/our-data https://dictionaryapi.com/products/json

Merriam Webster

Collegiate Dictionary API Documentation



2.10.10 PARENTHESIZED SENSE SEQUENCE: PSEQ

The parenthesized sense sequence groups together senses whose sense numbers form a sequence of parenthesized numbers.

Hierarchical Context

Occurs as an element in an sseq array.

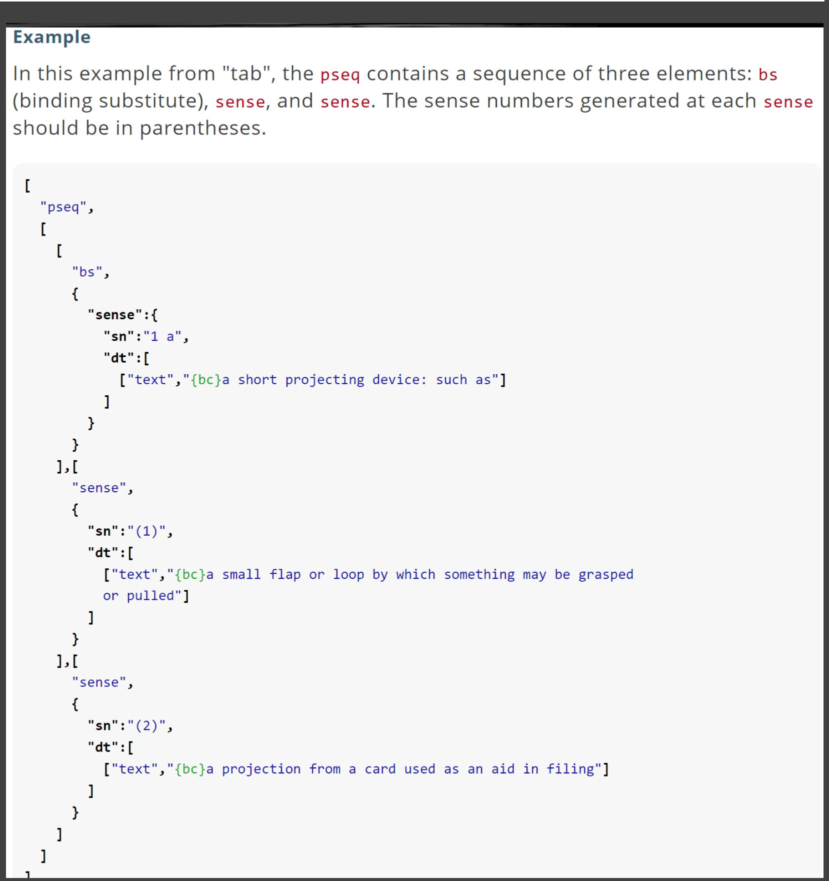
Display Guidance

If you are generating sense numbers for sense elements in a pseq sequence, put parentheses around the number. For example, the second sense in a sequence should have "(2)" as its sense number.

If you are instead using the sn to display the sense number, it will already contain the parentheses.

Data Model

array consisting of one or more sense elements and an optional bs element.



Example

In this example from "tab", the pseq contains a sequence of three elements: bs (binding substitute), sense, and sense. The sense numbers generated at each sense should be in parentheses.

```
[
  "pseq",
  [
    [
      "bs",
      {
        "sense":{
          "sn":"1 a",
          "dt":[
            ["text","{bc}a short projecting device: such as"]
          ]
        }
      }
    ],[
      "sense",
      {
        "sn":"(1)",
        "dt":[
          ["text","{bc}a small flap or loop by which something may be grasped
          or pulled"]
        ]
      }
    ],[
      "sense",
      {
        "sn":"(2)",
        "dt":[
          ["text","{bc}a projection from a card used as an aid in filing"]
        ]
      }
    ]
  ]
```