EDAM Studio Schema by Example

2023-02-26

# 1.0 Schema Inference by Example

In “EDAM Studio” (ES) Schemas are first class citizens if none is provided it will make the best effort to infer it given an example of a document instance. Today ES schema inference is only done on JSON documents but as time allows XML and others will be added. Internally ES don’t use any particular XML, JSON or other representation but its own flat-based agnostic schema representation of elements, and properties.

While using some of available JSON to JSON Schema tools, given the following example:

{

  "RVCT\_Document": {

    "Administration\_Information": {

      "Data\_Reported": "2023-02-27",

      "Date\_Counted": {

        "MMWR\_Week": 2,

        "MMWR\_Year": 3

      },

      "State\_Case\_Number": "String4",

      "Local\_Case\_Number": "String5",

      "Out\_Of\_State\_Case\_Reported\_Flag": "String6",

      "Out\_Of\_State\_Number": "String7",

      "Out\_Of\_Country\_Case\_Reported\_Flag": "String8",

      "Country\_Name": "String9"

    }

}

}

Will generate a property-based schema as follows:

{

"$schema": "http://json-schema.org/draft-XX/schema#",

"title": "some comment here",

"type": "object",

"properties": {

"RVCT\_Document": {

"type": "object",

"properties": {

"Administration\_Information\_": {

"type": "object",

"properties": {

"Data\_Reported": {

"type": "string"

},

"Date\_Counted": {

"type": "object",

"properties": {

"MMWR\_Week": {

"type": "integer"

},

"MMWR\_Year": {

"type": "integer"

}

},

"required": [

"MMWR\_Week",

"MMWR\_Year"

]

},

"State\_Case\_Number": {

"type": "string"

},

"Local\_Case\_Number": {

"type": "string"

},

"Out\_Of\_State\_Case\_Reported\_Flag": {

"type": "string"

},

"Out\_Of\_State\_Number": {

"type": "string"

},

"Out\_Of\_Country\_Case\_Reported\_Flag": {

"type": "string"

},

"Country\_Name": {

"type": "string"

}

},

"required": [

"Data\_Reported",

"Date\_Counted",

"State\_Case\_Number",

"Local\_Case\_Number",

"Out\_Of\_State\_Case\_Reported\_Flag",

"Out\_Of\_State\_Number",

"Out\_Of\_Country\_Case\_Reported\_Flag",

"Country\_Name"

]

}

},

"required": [

"Administration\_Information"

]

}

},

"required": [

"RVCT\_Document"

]

}

ES emphasis is on the definitions and will parse each Parent node as a “type” into something like:

{

"$schema": "http://json-schema.org/draft-XX/schema#",

"definitions": {

"RVCT\_Document\_": {

"type": "object",

"additionalProperties": false,

"properties": {

"Administration\_Information": {

"$ref": "#/definitions/Administration\_Information\_"

}

},

"required": [

"Administration\_Information"

],

"title": "RVCTDocument"

},

"Administration\_Information\_": {

"type": "object",

"additionalProperties": false,

"properties": {

"Data\_Reported": {

"type": "string",

"format": "date"

},

"Date\_Counted": {

"$ref": "#/definitions/Date\_Counted\_"

},

"State\_Case\_Number": {

"type": "string"

},

"Local\_Case\_Number": {

"type": "string"

},

"Out\_Of\_State\_Case\_Reported\_Flag": {

"type": "string"

},

"Out\_Of\_State\_Number": {

"type": "string"

},

"Out\_Of\_Country\_Case\_Reported\_Flag": {

"type": "string"

},

"Country\_Name": {

"type": "string"

}

},

"required": [

"Country\_Name",

"Data\_Reported",

"Date\_Counted",

"Local\_Case\_Number",

"Out\_Of\_Country\_Case\_Reported\_Flag",

"Out\_Of\_State\_Case\_Reported\_Flag",

"Out\_Of\_State\_Number",

"State\_Case\_Number"

],

"title": "Administration Information"

},

"Date\_Counted\_": {

"type": "object",

"additionalProperties": false,

"properties": {

"MMWR\_Week": {

"type": "integer"

},

"MMWR\_Year": {

"type": "integer"

}

},

"required": [

"MMWR\_Week",

"MMWR\_Year"

],

"title": "Date Counted"

}

},

"properties": {

"RVCT\_Document": {

"$ref": "#/definitions/RVCT\_Document\_"

}

}

}

Given an XSD, Database DDL, or other it will always catalog the Parents as types and its children as properties as done in a type system.

# 2.0 ES Extending Document Instances

As stated in the ES Mappings Specifications a source or target does not need to be fully defined but could be extended by adding additional types or elements that will eventually be part of the output. This is done within a Map Item Booklet (see 9999) as a code entry (for now) written in JSONata.

Using the RVCT\_Document fragment the following JSONata code:

{

   "Administration\_Information": {

       "Data\_Reported": RVCT\_Document.Administration\_Information.Data\_Reported,

       "Status": "Inactive"

   }

}

Will result in:

{

  "Administration\_Information": {

    "Data\_Reported": "2023-02-27",

    "Status": "Inactive"

  }

}

As shown a new element “Status” has been added and will be part of the output. ES will pick the new data element and will add it to the output as needed inferring (guessing) details about “Status”.

How to extend a source or target with specific information about this new “Status” element. Is this based on a code-set or be a string? and what is the specific definition for the element? To provide a way to improve on the specifics of a new element the “$definition” property is supported in JSON documents. For example, the JSONata request may look like:

{

   "Administration\_Information": {

      "Data\_Reported": RVCT\_Document.Administration\_Information.Data\_Reported,

      "Status": "Inactive"

   },

   "$definition": [

      {

         "name": "Status",

         "description": "Specify the status of administrative information"

      }

   ]

}

A “$definition” can state a single collection of properties as follows:

   "$definition": {

      "name": "Status",

      "description": "Specify the status of administrative information"

   }

Or an Array of those as follows:

   "$definition": [

      {

         "name": "Status",

         "description": "Specify the status of administrative information"

      }

   ]

Note that the “$definition” tag don’t change and there is no specific tag for a collection or a single item, it is expressed singularly that is the preferred way to reference to data elements and components shielding item reference from the specifics of cardinalities based on the schema definition rather than the tag names allowing the business to have needed flexibility to move from 0:1, 1:1 or 1:\* cardinalities anytime is needed.

ES will pick-up the definition(s) and extend the definition of the element with the provided information. At the end of the process definitions will not be part of the end result. Note that in a definition context (the JSONata) code fragment there are no ambiguities since there is only one “Status” element, in the case that an ambiguity may arise with the same name the above can be rewritten as:

{

   "Administration\_Information": {

       "Data\_Reported": RVCT\_Document.Administration\_Information.Data\_Reported,

       "Status": "Inactive",

       "$definition": [

          {

             "name": "Status",

             "description": "Specify the status of the provided administrative information"

          }

      ]

   }

}

The supported definition element properties include:

|  |  |
| --- | --- |
| name | The name of the element whose definition is been extended. |
| title | A short description of the property. |
| description | A narrative that describes the purpose of the element unambiguously. |
| path | The path of the element withing the context of the source or target. |
| side | Applicable to the “source”, “target”, or “both” side? (default: “target”) |
| element\_type |  |
| data\_type |  |