06/05/2022

Observation JSON Format

Présentation

Environmental Sensing

TABLE OF CONTENTS

1 INTRODUCTION

ObsJSON is a text format for the ES-Observation data.

This format is an application of the JSON format (RFC 8259), GeoJSON format (RFC 7946), Date and Time format (RFC 3339).

A binary version is also defined (Appendix) with CBOR format (RFC 8949)

1.1 CONVENTIONS USED

The 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.

The grammatical rules in this document are to be interpreted as described in [RFC5234].

1.2 TERMINOLOGY

The terms Json-Text, Json-Value (Value), Object, Member, Array, Number, String, False, Null, True are define in the JSON grammar.

The terms Geometry-type, Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon, GeometryCollection, GeoJSON-Types are defined in GeoJSON grammar.

Timestamp is defined in Date and Time format.

1.3 RULES

A Value in an ESValue SHOULD be unambiguous (i.e., parsers CAN deduce the Value type).

An ObsJSON-Text CAN contains all the ESObservation information (i.e., the ESObservation build from the ObsJSON-Text is identical to the initial ESObservation).

Values in Array are ordered and independent from the other Values.

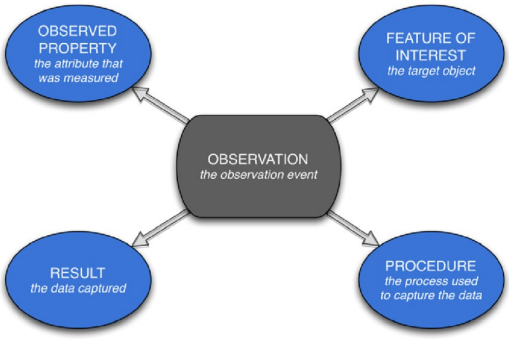
Members in Objects are not ordered.

2 ENVIRONMENTAL SENSING – OBSERVATION

2.1 PRINCIPLES

The concept of "Observation" is defined in the ISO19156 Standard. It allows to represent for example:

- Unit data from sensors,
- Modeling results,
- Geographical distributions,
- Temporal or trip histories,

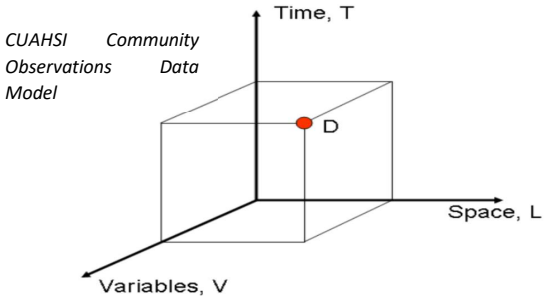


In this Standard, an Observation is characterized by:

- "Observed property": the observed property,
- "Feature of interest": the object (usually a place) of the observation,
- "Procedure": the information acquisition mode (sensor, model, etc.)
- "result": result of the observation or measurement

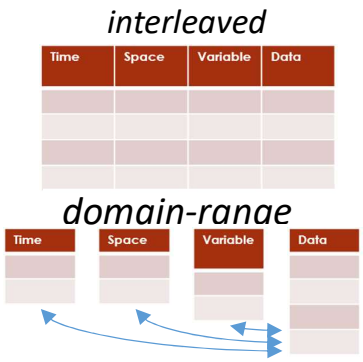
The result is a set of values referenced according to the 3 dimensions:

- Temporal,
- Spatial,
- Physical (observed property)



It can be converted into a 3-dimensional matrix, with each result indexed by temporal, spatial and physical values.

Note: This "domain range" indexed representation is preferred to an "interleaved" tabular representation which associates temporal, spatial and physical values with each result value.



Common properties (flags) are associated with each Observation. They make it possible to perform processing on the Observations without having to know their composition (e.g., bounding boxes, type of observation, volumetry, etc.).

Note: If the indexed representation is identical to the original, the "domain range" representation is equivalent to the "interleaved" tabular representation.

2.2 DATA MODEL

An Observation is an Ilist Object (Result, Features and Index data - see Appendix).

The Features included in the Ilist Object are "named values" (see chapter 4.8 ESValue) with several types: datation, location, properties.

Other information complete the Ilist Object :

- Attributes (name, id, type)
- Data (user data, information data, parameter data)

3 OBSJSON OBJECTS

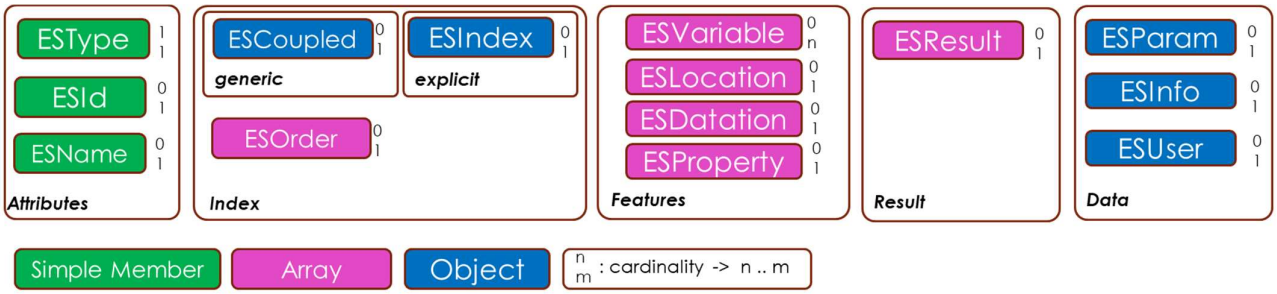
3.1 OBSERVATION

An ObsJSON is a JSON-text and represents an Observation.

ObsJSON consists of an object (ESObservation).

Description

The ESObservation Members are described in the figure below:



The Members are:

Category	Member	Key	Value
Attributes	ESName	« name »	String
Attributes	ESType	« type »	« observation »
Attributes	ESId	« id »	String
Result	ESResult	« result »	Array or ResultValue
Features	ESDatation	« datation »	Array or DatationValue
Features	ESLocation	« location »	Array or LocationValue
Features	ESProperty	« property »	Array or PropertyValue
Features	ESVariable	String	Array or Object
Index	ESIndex	« index »	Array or ESIdx
Index	ESCoupled	« coupled »	Object
Index	ESOrder	« order »	Array
Data	ESInformation	« information »	Object
Data	ESParameter	« parameter »	Object
Data	ESUserData	String	Object

Validity

An ESObservation is valid if:

- it contains at least the ESType Member,
- it contains at most one ESDatation, one ESLocation, one ESProperty, one ESResult Member

- each Member is valid.

Example

```
{“type”: “observation”, “datation”: “morning”, “location”: “paris”, “property”: “air quality”, “result”: “good”}

{“type”: “observation”, “datation”: “2021-01-05T22:18:26”, “location”: [2.4, 48.9], “property”: {“prp”: “PM10”, “unit”: “µg/m3”}, “result”: 51.3}
```

3.2 ATTRIBUTE MEMBER

3.2.1 ESType

Description

ESType is a single key/value Member:

- Key: “type”
- Value: “observation”

Validity

ESType is valid if key and value are as defined.

3.2.2 ESId

Description

ESId is a single key/value Member:

- Key: “id”

The value is a string (e.g., Database Id or filename)

Validity

ESId is valid if key and value are as defined.

3.2.3 ESName

Description

ESName is a single key/value Member:

- Key: “name”

The value is a string.

Validity

ESName is valid if key and value are as defined.

3.3 RESULT MEMBER

Description

Result is represented by an ESResult Member

Member	Key	Value
ESResult	« result »	Array of VariableValue

Note:

If the Array contains only one value, the square brackets MAY be omitted in the JSON String.

Validity

An ESResult Member is valid if:

- it contains at least one value
- each value is valid

Example

```
"result": 21.5                                                    one ResultValue
"result": [21.5, [2.4, 48.9], {"valid value": 43.3}, "high"]      four ResultValue
```

3.4 FEATURE MEMBER

Description

ESFeature is an Array. Value is specific for each ESFeature:

Member	Key	Value
ESDatation	« datation »	Array of DatationValue
ESLocation	« location »	Array of LocationValue
ESProperty	« property »	Array of PropertyValue
ESVariable	String	Array of ESValue

Note:

If the Array contains only one value, the square brackets MAY be omitted in the JSON String.

The value of ESVariable can be any ESValue.

Validity

An ESFeature Member is valid if:

- it contains at least one value
- each value is valid
- the Type of each value is consistent with the ESFeature

Example

```
"location": "paris"                                              one ESValue
"location": [[2.4, 48.9], [4.8, 45.8], [5.4, 43.3]]             three ESValue
"location": ["paris", "lyon", "marseille"]                       three ESValue
```

3.5 INDEX MEMBER

3.5.1 ESOrder

Description

ESOrder define the order in ESIndex (explicit index form) or in Result (generic index form). ESOrder is a String Array.

The strings in the Array are the ESFeature keys ordered according to indexing.

If ESOrder is not present, the default value ["datation", "location", "property", "first ESVariable key"] is used.

Validity

ESOrder is valid if:

- The values are present in ESFeature list,
- The length of the array equals the number of ESFeature.

Example

```
"order": ["datation", "property", "location"}
```

3.5.2 ESIndex

Description

ESIndex represent the "explicit form" of the index (see Indexed List Appendix)

It's a two-dimensional array following the order defined in ESOrder:

- One row for each ESFeature,
- One column for each ResultValue

The values in Array are integer.

Note:

If the ESObservation contains only one ESFeature, the square brackets for row MAY be omitted.
If the ESObservation contains only one ResultValue, the square brackets for column MAY be omitted.
If only one level of square brackets is present, the parser must decide if columns or rows are present

Validity

An ESIndex Value is valid if:

- it contains at least one integer value
- the number of rows equals the number of ESFeature
- the number of columns equals the number of ResultValue
- the integer values are positive and lower than the length of ESFeature

Example

```
“index”: [[0,2,1], [0,0,0]]            two ESFeature, three ResultValue
“index”: [0,2,1]                one ESFeature, three ResultValue (or the opposite)
```

3.5.3 ESCoupled

Description

ESCoupled is used only with the “generic form”.

ESCoupled is an Object. The members are key/value and represent coupled ESFeature:

- Key: ESFeature key
- Value: ESFeature key

If no one ESFeature is coupled, ESCoupled is not present.

Validity

ESCoupled is valid if at least one member is present and if ESFeature keys are present in ESObservation.

Example

```
“coupled”: {“datation”: “location”}                datation and location are coupled
```

3.6 DATA MEMBER

Data are Members where the Value MAY be an Object.

3.6.1 ESInformation

Description

The Value of an ESInformation is an Object where Members are:

Member	Key	Value
ObservationType	« typeobs »	String
LocationType	« typeloc »	String
DatationType	« typedat »	String
PropertyType	« typeprp »	String
ResultType	« typeres »	String
nValLocation	« nvalloc »	Integer
nValDatation	« nvaldat »	Integer
nValProperty	« nvalprp »	Integer
nValResult	« nvalres »	Integer
BoundingBox	« bbox »	Array (4 Float)
IntervalBox	« tbox »	Array (2 String)
Complet	« complete »	True / false
Score	« score »	Integer
Rate	« rate »	Float
Dimension	« dimension »	Integer
Axes	« axes »	Array (1 to 3 integers)

Validity

An ESInformation is valid if the Value contains at least one Member.

All the Value Members are optional.

Example

```
{"typeobs": "areaObsrecord"}                                          Minimal Value
{"typeobs": "areaObsrecord", "complete": false, "score": 226}     Defined Member
```

Note:

That information come from the other ESObervation elements.

A parser MAY ignore The ESInformation Member to build an ESObservation.

3.6.2 ESParameter

Description

The Value of an ESParameter is an Object where Members are:

Member	Key	Value
Reference	« reference »	String

ResultTime	« resulttime »	Timestamp or DateTime
PropertyDict	« pdict »	String
UniqueIndex	« unicindex »	True/false

Validity

An ESParameter is valid if the Value contains at least one Member.

All the Value Members are optional.

Example

```
{“unicindex”: true, “approbation”: true}
```

3.6.3 ESUserData

The structure of ESUserData Member is totally free.

The keys used in ESUserData MUST be different from those defined in the Reserved list name (see Appendix.)

4 ESVALUE

The ESValue are the Values included in the Feature or Result Arrays.

4.1 STRUCTURE

Description

An ESValue contains three information: A Name, a Value, a Type

One of the five formats SHALL be used for ESValue (where Name is a String):

- Type-Value format: {Type: Value} *(only for VariableValue)*
- Name-Value format: {Name: Value}
- Value format: Value
- Name format: Name *(not for VariableValue)*

The first format is used only with ESVariable or ESResult.

Validity

An ESValue Member is valid if:

- One of the four formats is used,
- it contains at least a Value or a Name

- each Value is valid compared to ESFeature or ESResult

Note:

The Name string MAY be used to represent:

- *detailed information (e.g., "beginning of the observation"),*
- *link to external information (e.g., "https://loco-philippe.github.io/ES.html"),*
- *id to link internal information (e.g., "res003" where "res003" is a key in a ESData Object),*

If the Member is an Object, the parser checks first if the key is a Type value.

If the Member is an Object and contains more than one Member, the Value Format is used.

Example

`"morning"`	*Name format*
`{"morning": "2021-01-05T10:00:00"}`	*Object format*
`[["2021-01-05T08:00:00", "2021-01-05T12:00:00"]]`	*Value format*

4.2 LOCATIONVALUE

Description

The Value of a LocationValue is a representation of a Point or a Polygon. It is defined by a Coordinates Array (as specified in GeoJSON).

Validity

A Value is valid if the Coordinates Array is valid and represents a Point or a Polygon.

Value example

`[2.4, 48.9]`	*Point*
`[[[2.4, 48.9], [4.8, 45.8], [5.4, 43.3], [2.4, 48.9]]]`	*Polygon*
`[[[0,0], [0,5], [5,5], [0,0]], [[1,1], [1,2], [2,2], [1,1]]]`	*Polygon with a hole*

Note:

The other Geometry-type are not allowed because the Coordinates Array is ambiguous:

- *LineString, Multipoint and Array of Point have the same representation*
- *Polygon and Array of LineString have the same representation*
- *MultiPolygon and Array of Polygon have the same representation*

The LineString in a Polygon MAY be open (without the last Point)

4.3 DATATIONVALUE

Description

A Date is defined by a String Timestamp (as specified in RFC 3339).

The Value of a DatationValue is a representation of a single Date or a Slot (MultiInterval):

- Date: String
- Slot: Array of one or multiple Interval (an Interval is an Array of two Date)

Validity

A Value is valid if the Date or Slot is valid.

Value example

```
"2021-01-05T10:00:00"                                   Date
[["2021-01-05T08:00:00", "2021-01-05T12:00:00"]]        Interval
[["2021-01-05", "2021-01-10"], ["2021-01-20", "2021-01-25"]]   Slot
```

Note:

Intervals MUST be represented by a Slot to avoid ambiguities with an array of Dates

If the DatationValue consists of a unique String, and if the String represents a Date, the parser SHALL assign the String to the Date, otherwise to the Name.

4.4 PROPERTYVALUE

Description

The Value of a PropertyValue is an Object. The Members are:

Member	Key	Value
PropertyType	« prp »	String (mandatory)
Unit	« unit »	String
SamplingFunction	« sampling »	String
Application	« application »	String
SensorType	« sensor »	String
UpperValue	« uppervalue »	Float
LowerValue	« lowervalue »	Float
Period	« period »	Float
UpdateInterval	« updateinterval »	Float
Uncertainty	« uncertainty »	Float

Note:

If the PropertyValue consists of a single Member (the PropertyType), it's not allowed to replace the Object by a string (the PropertyType value).
If the ESAtt PropertyDict is not defined, the default PropertyDict is used

Validity

A Value is valid if it contains at least the PropertyType ESAtt.

The PropertyType value MAY be present in a propertyDict how's define the Unit value

Example

`{"prp": "Temp"}`	*Minimal Value*
`{"prp": "Temp", "unit": "°c"}`	*Defined Member*
`{"prp": "Temp", "unit": "°c", "operation": "phase 1"}`	*User Member*

Note:

UserAtt MAY be used in the PropertyValue

For PropertyValue, the Name format is allowed (i.e., if the PropertyValue consists of a single string, this SHOULD be interpreted as the name).

4.5 VARIABLEVALUE

Description

The Value of a VariableValue CAN be any JSON Object.

Note:

For VariableValue, the Name format is not allowed (i.e., if the VariableValue consists of a single string, this SHOULD be interpreted as a Value).
If the Type of the VariableValue is the default type (standard value), il MAY be omitted.

Validity

Only the ESValue Rules.

Example

`21.8`	*Value format*
`{"low temperature": 2.4}`	*Object format*
`"https://loco-philippe.github.io/ES.html"`	*Value format*
`[21.8, {"test": true}]`	*Value format*

Note:

If the ResultValue is composed by a unique String, the parser SHALL assign the String to the Result.

5 APPENDIX : RESERVED VALUES

« type »
« id »
« datation »
« location »
« property »
« result »
« information »
« parameter »
« observation »
« prp »
« unit »
« sampling »
« application »
« sensor »
« uppervalue »
« lowervalue »
« period »
« updateinterval »
« uncertainty »
« typeobs »
« typeloc »
« typedat »
« typeprp »
« typeres »
« nvalloc »
« nvaldat »
« nvalprp »
« nvalres »
« bbox »
« tbox »
« complet »
« score »
« rate »
« dimension »
« axes »
« reference »
« resulttime »
« order »
« propdict »
« unicindex »

6 APPENDIX : EXAMPLES

- {“type”: “observation”, “datation”: “morning”, “location”: “paris”, “property”: “air quality”, “result”: “good”}

- {“type”: “observation”, “datation”: “2021-01-05T22:18:26”, “location”: [2.4, 48.9], “property”: {“prp”: “PM10”}, “result”: 51.3}

- {“type”: “observation”, “datation”: [“2021-01-05T22:18:26”, “2021-01-05T22:18:26”], “property”: [“air quality PM10”, “air quality PM2.5”], “result”: [10.2, 21.5, 51.3, 48]}

- {“type”: “observation”, “name”: “example4”, “id”: “example4.obs”,

  “parameter”: {“pdict”: “official”, “example”:4},

  “datation”:

  [“2021-01-04T10:00:00”,[[“2021-01-05T08:00:00”,“2021-01-05T12:00:00”]]],

  “location”:

  [[2.4, 48.9], [[[2.4, 48.9], [4.8, 45.8], [5.4, 43.3], [2.4, 48.9]]]],

  “property”:

  [{“prp”: “PM10”, “unit”: “µg/m3”}, {“prp”: “Temp”, “unit”: “°c”}]

  “result”: [51.3, {“low temperature”: 2.4}, 20.8, “high temperature”]

  “coupled”: {“datation”: “location”}}

  Note: another solution is to include index instead of “coupled”:

  “index”: [[0,0,1,1], [0,0,1,1], [0,1,0,1]]

7 APPENDIX : CBOR FORMAT

The Concise Binary Object Representation (CBOR – RFC8949) is a data format whose design goals include the possibility of extremely small code size, small message size, and extensibility without the need for version negotiation.

CBOR is based on the JSON data model: numbers, strings, arrays, maps (called objects in JSON), and a few values such as false, true, and null.

The CBOR format can be used with different options to minimize length:

- The precision of float values is adjustable from half precision (two bytes) to double precision (eight bytes),
- The datetime can be described by a standard text string (RFC3339) or by a numerical value (Epoch-based: six bytes).
- The ESFeature name can be represented with code value instead of string value
- The coordinates value can be described with integer instead of float (val_int = round(val_float)*10**7 : four bytes).

Example (Json format):

```
{"type": "observation",

"datation":["2021-01-04T10:00:00",[["2021-01-05T08:00:00","2021-01-5T12:00:00"]]],

"location":[[2.4123456,  48.9123456],  [[[2.4123456,  48.9123456],  [4.8123456, 45.8123456], [5.4123456, 43.3123456], [2.4123456, 48.9123456]]]],

"property": [{"prp": "PM10"}, {"prp": "Temp"}],

"result": [51.348, {"low": 2.457}, 20.88, "high"],

"coupled": {"datation": "location"}}
```

Example optimized (Cbor format):

```
{0 : [0,1,2],

1 : [[dt(2021, 1, 4, 10),[[dt(2021,1,5,8), dt(2021, 1, 5, 12]]],

     [[2.4123456, 48.9123456], [[[2.4123456, 48.9123456], [4.8123456, 45.8123456], [5.4123456, 43.3123456], [2.4123456, 48.9123456]]]],

     [{"prp": "PM10"}, {"prp": "Temp"}] ],

2 : [51.34375, {"low": 2.45703125}, 20.875, "high"],

3 : {0: 1} }
```

With :

- *Observation key codification:*
  - *0: "order"*
  - *1: "features"*
  - *2: "result"*
  - *3: "coupled"*
- *Order and coupled value codification:*
  - *0: "datation"*
  - *1: "location"*
  - *2: "property"*
- *Datation value: timestamp format*
- *Location value: integer representation (four bytes)*
- *Result value: half precision (two bytes)*

Length (bytes):

- JSON: 388
- CBOR: 298
- CBOR optimized: 133

8 APPENDIX : INDEXED LIST

An Indexed List (Ilist) represents an information (result) and the characteristics of this information (features).

Example: Observation, Measurement, Log…

An Ilist is an object with three components:

- Result: a named and ordered data set (array). The result elements can be every kind of object,
- FeatureSet: an ordered set of Features. Each Feature is a named and ordered set of every kind of object,
- Index: a specific object that describe the relationship between Result and Features

The main feature of this object is that there is no data duplication.

Example :

First name	*height*
Philip	*1.75*
Anne	*1.80*

Result : { 'height' : [1.75, 1.8] }

FeatureSet : { 'first name' : ['philip', 'anne'] }

Index : [[0, 1], [0, 1]]

All tabular data can be represented by an indexed list.

Example :

First name	*Last name*	*Weight*	*Height*
Philip	*Red*	*85*	*1.75*
Anne	*White*	*70*	*1.80*

Equivalent to:

First name	*Last name*	*Measure*	*Value*
Philip	*Red*	*Weight*	*85*
Philip	*Red*	*Height*	*1.75*
Anne	*White*	*Weight*	*70*
Anne	*White*	*Height*	*1.80*

Result : { 'value' : [85, 1.75, 70, 1.80] }

FeatureSet :
{ 'first name' : ['philip', 'anne'],
'last name' : ['red, 'white'],
'measure': ['weight', 'height'] }

Index :
Explicit: [[0, 0, 0], [0, 0, 1], [1, 1, 0], [1, 1, 1]]
or Generic: { 'coupled' : {0:1} }

The index can take several forms:

- Explicit form: For each Feature element, a number indicate the index of the result value (see examples above),

- Generic form: For each FeatureSet, two numbers can be defined :
  - Order: the result ordered priority of the FeatureSet
  - Coupled: the equivalent index FeatureSet
- Implicit form: The order is implicit if it is the same as the Feature order in the FeatureSet. The Reference is implicit if there is no equivalence.

In the last example, the generic form is :

- Coupled: {0: 1} (the index last name is coupled to the index first name)
- Order: [0, 1, 2] (the result is ordered first with name and second with measure). In this case, the order is implicit.

In most cases, the Index can remain implicit.

This representation of data is optimized (no duplication). The "cost" of added data (Index value) is minimal in the most cases (generic or implicit index).

Note: In a few cases, the implicit or the generic forms are not possible. It is then necessary to complete the data to obtain a complete set (or to have "trivial" index : [0, 1, 2, 3, …]),

Example:

First name	*Measure*	*Value*
Philip	*Weight*	*85*
Philip	*Height*	*1.75*
Anne	*Weight*	*70*

Result : { 'value' : [85, 1.75, 70] }
FeatureSet :
{ 'first name' : ['philip', 'anne'],
'measure': ['weight', 'height'] }
Index : [[0, 0], [0, 1], [1, 0]]

It can be automatically completed:

First name	*Measure*	*Value*
Philip	*Weight*	*85*
Philip	*Height*	*1.75*
Anne	*Weight*	*70*
Anne	*Height*	*-*

Result : { 'value' : [85, 1.75, 70] }
FeatureSet :
{ 'first name' : ['philip', 'anne'],
'measure': ['weight', 'height'] }
Index : implicit

With trivial index, the FeatureSet will be : { 'first name' : ['philip', 'philip', 'anne'], 'measure': ['weight', 'height', 'weight'] and the index is implicit.

Observation example: