# 8.3.5 Object representations used by the REST+JSON platform

## 8.3.5.1 Background (non normative)

The JSON and XML languages were designed for different purposes. XML is a meta-language and has a complementary XSD standard used to represent the format of XML documents. In contrast JSON is a language to represent data and lacks a universally recognized standard “JSON schema” language. However there are some proposed schema notations, including an (expired) IETF JSON Schema RFC [See: https://tools.ietf.org/html/draft-zyp-json-schema-04].

Mappings between XML and JSON are conventionally defined as a set of transformation rules that can be applied to a XML document resulting in a corresponding JSON document. Multiple mappings exist, each represents a different tradeoff and currently none is considered a “universal standard”.

The main issue in defining a XML to JSON mapping is that the transformation rules that result in the most “natural” or “comfortable” JSON formats have limitations, such as not being reversible back to the original XML, not preserving element order, not being able to handle repeated elements of different names, etc. In contrast the reversible transformations that can handle any document format result in unnatural-looking JSON that is hard to write and use in JavaScript programs.

The Web-Enabled DDS REST+JSON Platform definition requires only the ability to map a subset of XML, namely that which conforms to the XSD schemas referenced in the DDS REST+XML platform. This is a more constrained problem that a general XML to JSON mapping. The knowledge of these schemas can inform the transformation to improve the usability of the resulting JSON objects.

To allow XML to JSON transformations to be informed by the knowledge of the XSD that types the valid XML inputs to the transformation this specification defines a new new XML to JSON mapping called “Cuttlefish.” The Cuttlefish mapping can be configured to apply different rules on the XML element being transformed which is informed by the knowledge the XSD schema. This ability to “adapt itself” to the application context inspired the name “Cuttlefish”.

## 8.3.5.2 Object representations in JSON

The JSON representation of all the parameters and return values used in the REST+JSON platform is obtained by applying the JSON to XML transformation rules defined below. The collection of this transformation rules is referred to as the “Cuttlefish” XML to JSON mapping.

The Cuttlefish transformation is aware of the XSD schemas that define the documents that can be inputs to the transformation. The transformation rules can be customized based on the specific XML element being transformed. From analyzing the XSD the **element\_repetition\_kind** and **element\_content\_kind** can be determined:

|  |  |  |
| --- | --- | --- |
| content\_kind | Constraints (per XSD) | Examples |
| PRIMITIVE\_TEXT | Element can only hold text. It can have no attributes or children other than the text | <elemTextElem>Hello</elemTextElem> |
| PRIMITIVE\_NUMBER | Element can only hold a number. It can have no attributes or children other than a number | <primNumElem>12</primNumElem>  <primNumElem>-1.2</primNumElem>  <primNumElem>1e-12</primNumElem> |
| PRIMITIVE\_BOOLEAN | Element can only hold two values logically equivalent to ‘true’ and ‘false’ | <primNumElem>true</primNumElem> |
| ARRAY | - Element has no attributes  - Element has no text, the children are all elements themselves  - All children have the same name  - The children have no attributes | <arrayElem>  <item>4</item>  <item>6</item>  </arrayElem>  <arrayElem>  <item>  <x>5</x> <y>5</y>  </item>  <item>  <x>7</x> <y>9</y>  </item>  </arrayElem> |
| GENERIC | Element does not fall into the previous categories | <genericElem attr=”1>Value</genericElem>  <genericElem>  <x>5</x> <y>5</y>  </genericElem> |

|  |  |  |
| --- | --- | --- |
| repetition\_kind | Meaning | Examples |
| NOT\_REPEATED | Element cannot appear repeated within parent element | <parentElem>  <notRepeated1>a</notRepeated1>  <notRepeated2>b</notRepeated2>  </parentElem> |
| SINGLE | Element can appeared repeated but all its siblings have the same name | <parentElem>  <repeatedSingle>a</repeatedSingle>  <repeatedSingle>a</repeatedSingle>  </parentElem> |
| MIXED | Element can appear repeated and there can be siblings with different names | <parentElem>  <repeatedMixed>a</repeatedMixed>  <repeatedMixed>b</repeatedMixed>  <otherSibling>c</otherSibling>  </parentElem> |

XSD The customization is done by setting the value of several parameters logically associated with each XML element. These parameters are: “**key\_attribute**”, “**element\_name\_property**”, “**shortcut\_attribute**”, “**grouping\_style**”.

|  |  |
| --- | --- |
| Parameter | Meaning |
| key\_attribute | Name of a non-optional element attribute that can be used to uniquely identify the element within its parent.  If this is defined for an element, then it must be defined for all other sibling elements. The scope of the “unique identification” is across all siblings of the same parent |
| element\_name\_property | Property name to be used to store the element name within the resulting JSON object. This is significant only if the key\_attribute is defined.  If the key\_attribute is defined and the element\_name\_property is not defined then the name of the element will not appear explicitly in the resulting JSON document. |
| shortcut\_attribute | Name of a element attribute that represents the most common use of the element. If an element defines a shortcut\_attribute it indicates that when the element contains no additional information it can be represented using only the value of the shortcut\_attribute. |
| sibling\_ordering | One of NONE, ORDERED\_BY\_ELEMENT, ORDERED\_ALL  NONE indicates that the order of an element relative to its siblings is not significant  ELEMENT\_ORDERED indicates that the order is significant relative to other siblings with the same element name but not relative to siblings with a different element name.    ORDERED\_ALL indicates ordered is significant relative to siblings whether they have the same name or not |
| sibling\_grouping | Applies to elements with repetition\_kind different from groupl.  It can be one of NONE, GROUPED  GROUPED indicates siblings with the same name should be grouped together.  NONE indicates siblings should not be grouped . |

When Cuttlefish is used to transform from the XML documents that appear in the REST+XML platform to the corresponding JSON data used by the REST+JSON the customization parameters are defined as indicated in the table below.

|  |  |
| --- | --- |
| ***XML Element*** | ***Cuttlefish parameters*** |
| Root element with name **dds** | key\_attribute → Not Set  element\_name\_property → N/A  shortcut\_attribute → Not Set  sibling\_ordering → NONE  sibling\_grouping → NONE  // Information from XSD  content\_kind → GENERIC  repetition\_kind → MIXED |
| Elements with name one of:  **module, struct, valuetype, union, enum, sequence, typedef, const, forward\_decl, bitset, annotation**  who are children of **dds.types** | key\_attribute → “name”  element\_name\_property → “@kind”  shortcut\_attribute → Not Set  sibling\_ordering → NONE  sibling\_grouping → NONE  // Information from XSD  content\_kind → GENERIC  repetition\_kind → MIXED |
| Elements with name one of:  **struct.member,**  **valuetype.member,**  **annotation.member,**  **union.case.member,**  **enum.enumerator**  who are children of **dds.types** | key\_attribute → “name”  element\_name\_property → Not Set  shortcut\_attribute → **“type”**  sibling\_ordering → ORDERED\_BY\_ELEMENT  sibling\_grouping → GROUPED  // Information from XSD  content\_kind → GENERIC  repetition\_kind → SINGLE |
| Elements with name one of:  **dds.qos\_library,**  **dds.domain\_participant\_library,**  **dds.application,**  **dds.waitset\_library,**  **dds.dataset\_library** | key\_attribute → “name”  element\_name\_property → Not Set  shortcut\_attribute → **Not Set**  sibling\_ordering → NONE  sibling\_grouping → GROUPED  // Information from XSD  content\_kind → GENERIC  repetition\_kind → SINGLE |
| Elements with name one of:  r**egister\_type, topic,**  **publisher, subscriber, data\_writer, data\_reader**  who are children of either  **dds.domain\_participant\_library.domain\_participant** or **dds.application.domain\_participant** | key\_attribute → “name”  element\_name\_property → Not Set  shortcut\_attribute → **Not Set**  sibling\_ordering → NONE  sibling\_grouping → GROUPED  // Information from XSD  content\_kind → GENERIC  repetition\_kind → MIXED |
| Elements with name one of:  **dds.qos\_library.qos\_profile,**  **dds.domain\_participant\_library.domain\_participant,**  **dds.application.domain\_participant,**  **dds.waitset\_library.waitset,**  **dds.dataset\_library.dataset** | key\_attribute → “name”  element\_name\_property → Not Set  shortcut\_attribute → Not Set  sibling\_ordering → NONE  sibling\_grouping → NONE  // Information from XSD  content\_kind → GENERIC  repetition\_kind → SINGLE |
| Elements with name one of:  **dds.dataset\_library\_dataset.readSampleSeq,**  **dds.dataset\_library\_dataset.writeSampleSeq**  And also elements with name one of:  **sample.data.item**  who are children of either  **dds.dataset\_library\_dataset.readSampleSeq** or **dds.dataset\_library\_dataset.readSampleSeq**  And also elements with name one of:  **subscriber.data\_reader.filter.parameter\_list**  who are children of either  **dds.domain\_participant\_library.domain\_participant** or **dds.application.domain\_participant**  And also elements with name:  **publisher\_qos.partition.name** or  **subscriber\_qos.partition.name**  who are children of either  **dds.qos\_library.qos\_profile** or  **dds.domain\_participant\_library.domain\_participant.publ sher** or  **dds.domain\_participant\_library.domain\_participant.subscriber** or  **dds.application.domain\_participant.publisher** or  **dds.application.domain\_participant.subscriber** | key\_attribute → Not Set  element\_name\_property → N/A  shortcut\_attribute → Not Set  sibling\_ordering → NONE  sibling\_grouping → NONE  // Information from XSD  content\_kind → ARRAY  repetition\_kind → SINGLE |
| All elements not listed above and elements newly introduced as part of the transformation process | key\_attribute → Not Set  preserve\_elemname → N/A  shortcut\_attribute → Not Set  sibling\_ordering → NONE  sibling\_grouping → NONE  /Information from XSD  content\_kind → GENERIC or PRIMITIVE per the XSD definition  repetition\_kind → NOT\_REPEATED |

### 8.3.5.2.1 Cuttlefish rules

A JSON data value can be an Object, Array, Number, String, or the special values null, true, and false.

JSON objects contain zero or more name/value pairs. A name/value pair is also called a property. Objects are represented enclosed by curly braces where each property name is separated from its value by a “:” and successive properties separated by a “,”.

{

<property\_name1> : <pvalue1>,

<property\_name2> : <pvalue2>

}

The <property\_name> is always a string and the <pvalue> can be any JSON value.

JSON arrays contain zero or more JSON values. Arrays are represented enclosed by square braces where each successive value is separated by a “,”.

[ <pvalue1>, <pvalue2> ]

The transformation of a XML document to a JSON document is performed in two steps. First the XML document is prepared and second a set of rules are applied.

XML preparation.

1. The XML is normalized. All element text is trimmed and comments are discarded.
2. XML element namespaces are discarded.
3. Elements with repetition\_kind=MIXED and sibling\_grouping=GROUPED are reordered such that they elements with the same name appear next to each other. The reordering is done in a way that preserves the relative order among member with the same element name.
4. Elements with sibling\_grouping=GROUPED are replaced within their parent with a single element with name matching the original name and the suffix “\_list” appended. The original elements become children of this new element. The

Transformation rules:

1. The root element is transformed into a JSON object with a single property with propertyKey set to the root element name and property\_value an JSON object.
2. XML Elements are mapped to either JSON values or JSON properties:
   1. Children elements whose parent property pvalue (or value pvalue) is a JSON Object become properties in the parent’s pvalue Object. Each child becomes a separate JSON property within the parent object.
   2. Children elements whose parent *pvalue* is a JSON Array become JSON values within the parent array. Each child becomes a separate value. The value representing the child may be a JSON object, string, or a number per application of the rules that follow.
3. If a child is mapped to a property ( <property\_name> : <pvalue> ) , the property\_name is
   1. The name of the element if **key\_attribute** is not defined
   2. The value of the attribute named **key\_attribute** if it is defined.
4. If the child is mapped into a property, the *property\_value* is either:
   1. A primitive value if **content\_kind** is PRIMITIVE\_TEXT, PRIMITIVE\_NUMBER or PRIMITIVE\_BOOLEAN. In this case the corresponding JSON value is used.
   2. A JSON array if **content\_kind**=ARRAY or if any children have a sibling ordering different from NONE.
      1. The element children appear as values within the array, one value per child, in the same order as they appear in the XML document.
      2. The attributes appear grouped as a single JSON as the last element in the array. The JSON object contains a single property named “@attributes” with an JSON object as value. The value object contains properties , one per attribute, the property name is the attribute name preceded by “@” and the property value a JSON primitive type matching the attribute value.
   3. A primitive value if the **content\_kind** is GENERIC, the element has defined a **shortcut\_attribute**, and the element contains no other attributes or elements other than the **shortcut\_attribute** and the **key\_attribute** (if defined).
   4. A JSON object if none of the above is met.
      1. All element attributes, except the **key\_attribute** (if defined) become properties in the JSON object. The property name is the same as the attribute name preceded by a “@”. The property value is either Number, Boolean, or String according to the attribute value.
      2. If the **element\_name\_property** is defined, then there an additional property with name the value of element\_name\_property is added to the JSON object. The value of this property is a string containing the element name.
5. If the child is mapped into JSON value, the value is either a JSON primitive value or an JSON object.
   1. If the element has **key\_attribute** defined element is mapped to a JSON object containing a single property constructed applying rules 3.2 and 4 above.
   2. Otherwise the element is mapped to a value applying the same rules as if we were constructing the *property\_value* for that element using the rules in 4 above.
   3. As consequence of the above the element name does not appear in the resulting JSON value unless both **key\_attribute** and the **element\_name\_property** are defined.

See Annex A for an example of how these rules apply to the types of XML documents used in this specification.