OAGi
Open Applications Group

# Specification for Serialization of BIE to JSON Schema

Version 1.0

Published 2019-10-30

## Credits

Editor: Serm Kulvatunyou, NIST

Other Contributors:

Scott Nieman, Land O'Lakes

Steffen Fohn, ADP

Michael Rowell, Oracle

Jim Wilson, OAGi

Hakju Oh, NIST

Nikola Stojanovic, Invited Expert

Athanasios Dimitriad, NIST

Sofian Chouder, NIST

Nenad Ivezic, NIST

# Change Tracking

| Date | Editor | Version | Change Detail |
|---|---|---|---|
| 10-30-2019 | Serm Kulvatunyou | 1.0 | First version |
| | | | |
| | | | |
| | | | |
| | | | |

## Table of Contents

# 1 Glossary

**ABIE**

Aggregate Business Information Entity per CCS. It can also refer to the ABIE table in the OAGIS Repository.

**ACC**

Aggregate Core Component per CCS. It can also refer to the ACC table in the OAGIS Repository.

**ASBIE**

Association Business Information Entity per CCS. It can also refer to the ASBIE table in the OAGIS Repository.

**ASBIEP**

Association Property Business Information Entity per CCS. It can also refer to the ASBIEP table in the OAGIS Repository.

**ASCC**

Association Core Component per CCS. It can also refer to the ASCC table in the OAGIS Repository.

**ASCCP**

Association Core Component Property per CCS. It can also refer to the ASCCP table in the OAGIS Repository.

**BBIE**

Basic Business Information Entity per CCS. It can also refer to the BBIE table in the OAGIS Repository.

**BBIE_SC**

A table in OAGIS repository containing SC restrictions for a BBIE.

**BCC**

Basic Core Component per CCS. It can also refer to the BCC table in the OAGIS Repository.

**BDT**

Business Data Type per CCS

**BIE**

Business Information Entity per CCS. Conceptually BIE encompasses all types of business information entities defined below. Within the SRT, a BIE is a profile (contextualized, subset) of a CC, which is a component within the OAGIS model.

**BIE element**

A data element in a BIE. For example, if a BIE is the Address component, a BIE element can be an AddressLine, City, State, etc.

**CC**

Core Component per CCS. Conceptually CC encompasses all types of core components and BDT defined below. Within the SRT, the OAGIS model content is treated/stored as CCs.

**CCS**

UN/CEFACT Core Component Specification (formerly, CCTS, Core Component Technical Specification)

**Expression**

A syntax-specific representation of a specification (CC or BIE).

**GUID**

Globally Unique Identifier

**JSON or JSON instance**

An instantiation of a JSON schema. Note that a JSON schema is also a JSON instance.

**JSON schema**

A JSON document that conforms to JSON Schema. *Note lower-case "s". Often the distinction between "JSON schema" and "JSON Schema" can be correctly understood from the context, but sometimes the explicit distinction is important.*

**JSON Schema**

Note upper-case "s". JSON Schema specification, JSON Schema standard, or the JSON Schema standard per jsonschema.org. *See "JSON schema" note.*

**OAGIS component**

Any OAGIS BOD, Noun, Subcomponent of Noun, Common Component

**OAGIS Model**

The canonical representation of OAGIS where BOD and component definitions are represented with abstractions, reusable components, and full content.

**OAGIS Model JSON schemas**

This would be the OAGIS Model serialized in JSON schema syntax. This canonical JSON schemas currently does not exist in an OAGIS nor Score distribution. If needed, such schemas can be generated from the Score tool.

**OAGIS Model XML schemas**

This is OAGIS Model expressed/serialized in the Garden of Eden style XML Schema syntax. In other words, the OAGIS canonical XML schemas as defined in the OAGIS enterprise edition under the Model folder.

**OAGIS Repository**

OAGIS specifications in a database management system. It is part of the SRT.

**SC**

Supplementary Component (of a BDT) per CCS

**Score**

Score is a new name given to the SRT.

**Serialization**

Expression.

**SRT**

Semantic Refinement Tool. It is a tool for managing the lifecycle of CCs and BIEs.

**Standalone OAGIS JSON schemas**

The all-inclusive OAGIS JSON schemas of an OAGIS component with the same schema design pattern as that of the Standalone OAGIS XML schemas. Such schemas are currently not distributed as part of OAGIS. Conceptually, such schemas can be generated from Score.

**Standalone OAGIS XML schemas**

The all-inclusive OAGIS XML schemas of an OAGIS component. At present, standalone XML schemas are at the BOD level and are distributed in the Standalone folder of OAGIS enterprise and standard editions.

**Syntax independent [X]**

*Where "X" is "OAGIS Model", "Model BOD", "Model Component", "Standalone Component", or "BIE".* OAGIS Model, Model BOD, Model Component, Standalone Component, and BIE represented via Core Component Specification in the OAGIS Repository. It is important to note that the terms OAGIS Model, Model BOD, Standalone Component, and BIE are always used in this abstract, syntax-independent sense in this specification. To refer to a syntax-specific representation of these concepts, these terms are suffixed with the name of the particular syntax (expression), e.g., OAGIS Model JSON schema, BIE JSON schema, BIE XML schema.

# 2 Conventions

`Courier New Font Size 9`: Used for Code snippet, Element, Type, Entity name.

"Literal": Quotation around a string is used for a literal value, JSON key, or JSON key value.

*Italicize*: Text is italicized for emphasizing purpose.

# 3 Purpose

The purpose of this document is to describe the JSON schema convention to represent the BIE. The BIE can be any OAGIS component (including common component or subcomponent of a noun, noun, or BOD). The purpose of a BIE JSON schema is to provide precise semantic restrictions of a BIE usage information in a JSON schema while also simplify the schema structure as much as possible.

A BIE JSON schema is a standalone schema (in OAGIS lingo) where all the content is self-contained in a single schema file. Three important characteristics of the BIE JSON schema are as follows.

First, the BIE JSON schema employs, for the most part, the Russian Doll[1] XML schema like design. In Russian Doll, there is one root element (the BOD) and types are *anonymous-type*. In the BIE JSON schema, there is one root element and most types are anonymous except code list and primitive types.

The objective of the BIE JSON schema is to make the content model as precise as possible. Using the Russian Doll design allows the same types (e.g., supplier party type, business data types) used in multiple places within a single BIE to have different contents (or restrictions). With the BIE JSON schema design, integration interfaces will be able to declare precise integration requirements.

Second, the schema mainly conveys the data structure definition. It is a pure tree structure containing only data element nodes. Model features such as extension (inheritance), restriction, and group are removed and only their data element nodes remain.

The last key characteristic is that the BIE JSON schema implements the Core Component Specification (CCS) context mechanism. Therefore, the profile schema may contain data elements which are only subset of the model or traditional OAGIS standalone schema it corresponds to. In addition, it can contain meta-data (e.g., component identity and version information), *context* information, and context specific documentations. The context information and context specific information describes suitable usage information of the BIE.

# 4 Scope

A BIE can have many expressions such as XML Schema, JSON Schema, or others. The scope of this document is limited to the JSON Schema expression.

Below are requirements identified as out-of-scope as of the current version of this document.

## 4.1 Schema size optimization

The size of a plain Russian-Doll-based BIE JSON schema can be very large if it contains thousands of data fields because types are repeatedly defined even for the common attributes used throughout the schema. The schema can be made more compact by *globally* defining types that do not have different content or restriction. In other words, anonymous types that have a common content model can be globally defined and reused. For example, if there is no difference in the `SupplierParty` used in the header and the line of a BIE, only a single global SupplierPartyType needs to be defined. Similarly, if there is no different restriction between a `TaxAmount` and a `TotalAmount` field, a single global `AmountType` can be defined and reused. The actual BIE can be complicated to optimize than these two examples such as when there are a few Amount typed fields with one content model and there are other few Amount typed fields with another content model and so on.

At present, the working group member see no need for such optimization as BIEs are typically small. Therefore, the optimization to make the BIE most compact, yet precise, is not in scope of the current specification. The current specification only reduces the size of the profile BIE by using global types for code list definitions and for OAGIS-defined built-in

[1] http://www.oracle.com/technetwork/java/design-patterns-142138.html

primitive types (the naming convention, enforced in the Score Tool, makes the code list and built-in type names unique).

## 4.2 Advance semantic restriction

There are also requirements from the user community to represent more complicated semantic restrictions such as dependencies between the values of data elements. Although such capability does not exist in XML Schema, JSON Schema has provisions for that kind of restrictions/validations. However, this is out of scope for this version of the specification. We documented here some of the major requirements for consideration in the future versions of the specification.

1. To represent IF-THEN type of rules for conditional restrictions across data elements.
2. To represent specific values needed in a particular occurrences of a repeated data element, e.g., when two IDs of the Party element are needed; and the requirement is that one is the DUNS type of ID and the other is the government issued Tax ID.

## 4.3 XML-JSON round-tripping

The working group has attempted at creating a JSON schema that allows for round-tripping between XML and JSON syntaxes; however, there are a few major issues, particularly with the translation from JSON back to XML. One of the major issues is the loss of the sequence of elements in JSON. The other issue is that JSON does not distinguish element and attribute; every data is represented as a key and value. Additional meta-data, such as namespace, sequence, object type, need to be populated in the generated JSON or JSON schema in order that a round tripping is possible. The group did not see an immediate need for such a round-tripping; therefore, the requirement is considered out-of-scope in this version. Preferences were given to the simple and compact design.

## 4.4 Matching primitives with XML schema primitives

The serialization will use only JSON schema built-in type and its related constraint keywords (e.g., “format”, “multipleOf”) to represent the primitive types used in the OAGIS Model. Only the duration JSON Schema serialization use the pattern keyword. Therefore, the value space of the JSON Schema serialization does match that of the XML Schema serialization.

The pattern keyword in JSON schema that uses the regular expression has been tried to match the lexical and value spaces with the primitives used in the OAGIS model. However, more testing is needed. The current mapping between the primitives used in OAGIS model and JSON Schema primitives will be provided in section 8.6.

## 4.5 BIE meta-data

Most BIE meta-data such as business context, dictionary entry details, and based core component data are out of scope in this version.

## 4.6 OAGIS Model JSON schemas

Generating a set of JSON schemas that reflect the OAGIS model is out of scope in this document.

## 4.7 Full BIE JSON schema for importing/exporting including CC references

Generating full a BIE schema that may be used for export/import from/into an OAGIS repository is out of scope of the current version.

### 4.8 Open API Generation

The OAGi JSON/Mobile working group considers the functionality to generate Open API specification. However, this has been deferred to future effort.

## 5 Relevant Specifications

| | |
|---|---|
| Core Component Specification Version 3.0 | https://www.unece.org/cefact/codesfortrade/ccts_index.html |
| JSON Schema Core, Draft-05 | https://tools.ietf.org/html/draft-wright-json-schema-00 |
| JSON Schema Validation (for keywords), Draft-05 | https://tools.ietf.org/html/draft-wright-json-schema-validation-00 |
| Open API Specification (OAS) Initiative | https://github.com/OAI/OpenAPI-Specification/blob/master/README.md |
| OAS version 3.0.1 | https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.1.md |
| OAGIS Release 10.X Naming and Design Rules | http://www.oagi.org/oagi/downloads/ResourceDownloads/UNCEFACT_XML_NDR_V3p0.pdf |
| Overview of the OAGIS Repository, a Component of the Score Tool | https://oagi.org/OurCommunity/WorkingGroups/tabid/149/Default.aspx |
| OAGIS Repository Data Model | https://drive.google.com/open?id=1hqu7P7_fVQ4NXk6ZQgxMbH3qmha6gfXb |
| RFC 2119 | The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in Internet Engineering Task Force (IETF) Request For Comments (RFC) 2119. |

# 6 Tools Used for Creation of This Document

https://www.jsonschemavalidator.net/

# 7 Overall Design

The BIE JSON schema generally follows the OAGIS Release 10.X Naming and Design Rules (see Relevant Specification) except those described in this section. The design rules described in this section shall override relevant rules in the OAGIS Release 10.X Naming and Design Rules.

The overall schema design objective is for it to be simple/friendly yet provides precise semantic constraints. Therefore, the Russian Doll JSON schema design pattern is used and layers of type inheritances (extension and restriction) are reduced to a single type definition; and group wrappers exist in the OAGIS model are skipped. These overall design details are explained in this section.

## 7.1 Compatibility

A BIE JSON schema shall be compatible with the corresponding OAGIS Model JSON schema. In other words, JSON instances valid according to a BIE JSON schema shall be valid according to the corresponding Model JSON schema.

*Rule 1: JSON instances valid against a BIE JSON schema shall be valid against the corresponding element in the Model JSON schema.*

It should be noted that at this time the Model JSON schema only virtually exists. Rule 1 is programmatically compliant by the tool generating it.

## 7.2 JSON Schema Design Pattern

The BIE JSON schema employs the Russian Doll design pattern (i.e., local elements and anonymous types). **Only** code lists, agency ID lists, and primitives are defined globally and are reusable. Other specifics of this pattern are specified in section 8.

## 7.3 Plural

Although the API community commonly use plural terms for array, after several discussions in the OAGi JSON Mobile WG it was decided that pluralizing the name for JSON array would not be adopted. The reasons are 1) some English words, common acronyms or specific names (e.g., SCAC, UPC) do not have plural form and may cause confusion; and 2) OAGIS canonical names are in singular complying with UN/CEFACT NDR and automatically pluralizing these names are not reliable due to #1.

## 7.4 Lower Camel Case

Lower camel case is commonly use in JSON developer communities. The OAGi JSON mobile WG decided to follow that. See subsequent sections for more details on how OAGIS canonical names are automatically converted into lower Camel Case throughout JSON Schemas.

## 7.5 Meta Schema

A meta-schema must be declared according to Rule 2. Example 1 below shows the meta-schema declaration.

*Rule 2: Meta schema declaration shall be made at the top of JSON schema via the "$schema" key. It SHALL point to draft-04 meta-schema.*

```
{ "$schema": "http://json-schema.org/draft-04/schema#" }
```

*Example 1: Schema declaration*

## 7.6 Root Property Key

A BIE JSON schema generally contains only one root key representing the top-level ASBIEP the user selected for serialization. All other property keys and object definitions are local except the type definitions for primitives, code lists, and agency identifier lists used in the BIE. Top-level ASBIEP shall be generated according to Rule 3.

*Rule 3: A property named after the top-level ASBIEP selected for the schema serialization shall be specified at the root of the BIE JSON schema. Property term of the ASBIEP shall be used as the property name following the naming convention in Rule 4. The content model of the property shall be defined as a JSON object or an array of a JSON object following the content model of the top-level ABIE that is associated with the top-level ASBIEP. Whether JSON object or an array of a JSON object is generated is an option selected by the user on the SRT user interface.*

*Rule 4: Names of all generated JSON properties shall use lower-camel case. The term `Identifier` shall be kept as `Identifier` instead of using the `ID` abbreviation. A token in the Object Class Term, Property Term, Representation Term, or Data Type Term that is an acronym, i.e., spelled using all upper case in the OAGIS repository, shall be converted to camel case. For example, a property term "`UPC Code`" or "`Product UPC Code`" shall be generated in JSON schema as "`upcCode`" or "`productUpcCode`", respectively.*

*Rule 5: When the schema package option is not selected or when the schema package option is selected but there is only one top-level ASBIEP selected, each top-level ASBIEP shall be serialized into a separate schema file. In addition, the root schema shall have the "required" key instantiated reflecting that the corresponding JSON property is required. On the other hand, if the schema package option is selected and multiple top-level ASBIPs are selected, the root schema shall have multiple JSON properties corresponding to the top-level ASBIEPs serialized; and the "required" key shall not be instantiated.*

Example 2 below illustrates the "`showInspectionOrde`r" top-level ASBIEP expressed in JSON schema. In this example, the user did not select an array option; hence, the "type" of "`showInspectionOrder`" is simply an object (Example 3 shows the case when the array option is selected). The corresponding top-level ABIE has two required properties that are "`applicationArea`" and "`dataArea`" and an optional property that is "`systemEnvironmentCode`". The primitive "token", code list "`systemEnvironmentCode`", and agency identifier list "`schemeIdentifierList`" are defined globally in the reusable JSON definition block at the bottom.

```
{
        "$schema": "http://json-schema.org/draft-04/schema#",
        "required": ["showInspectionOrder"],
        "additionalProperties": false, //This is needed because the default is true.
        "properties": {
                "showInspectionOrder": {
                        "description": "Top-level ABIE context definition.",
                        "type": "object",
                        "required": ["applicationArea", "dataArea"],
                        "additionalProperties": false,
                        "properties": {
                                "systemEnvironmentCode": {
                                        "description": "BBIE context definition.",
                                        "type": "object",
                                        "required": ["content"],
                                        "additionalProperties": false,
                                        "properties": {
                                                "content": {
                                                        "$ref": "#/definitions/systemEnvironmentCodeList"
                                                },
                                                “listAgencyIdentifier”: {
                                                        “type”: “string”
                                                }
                                        }
                                },
                                "applicationArea": {
                                        "description": "ASBIE context definition.",
                                        "type": "object",
                                        //Further specification of the applicationArea property would be given here.
                                },
                                "dataArea": {
                                        "description": "ASBIE context definition.",
                                   "type": "object",
                                        //Further specification of the dataArea property would be given here.
                                }
          }
  }
        "definitions": {
                "schemeIdentifierList": {
                        "type":"string",
                        "enum": ["Internal", "DUNS", "GS1"]
                },
                "systemEnvironmentCodeList": {
                        "type":"string",
                        "enum": ["Test", "Production"]
                },
                "token": {
                        "type": "string"
                }
        }
}
```

*Example 2: Root property key with the regular object option*

```
{
	"$schema": "http://json-schema.org/draft-04/schema#",
	"required": ["showInspectionOrder"],
	"additionalProperties": false, //This is needed because the default is true.
	"properties": {
		"showInspectionOrder": {
			"description": "Top-level ABIE context definition.",
			"type": "array",
			“items”: {
				“type”: “object”,
				"required": [
					"applicationArea",
					"dataArea"
				],
				"additionalProperties": false,
				"properties": {
					"systemEnvironmentCode": {
						<!--Same as in Example 2 -->
						},
					"applicationArea": {
						<!--Same as in Example 2 -->
					},
					"dataArea": {
						<!--Same as in Example 2 -->
					}
				}
			}
		}
	}
	"definitions": {
		<!--Same as in Example 2 -->
	}
}
```

*Example 3: Root property key with array option*

Schema package is commonly used for supporting Web Services deployment. Example 4 below illustrates the case when multiple top-level ASBIEPs including `“bom”`, `“workOrderHeader”`, and `“workOrder”` are selected and the schema package option is enabled.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": false,
  "properties": {
    "bom": {
      "type": "object",
      <!-- Content model of the BOM goes here. -->
    },
    "workOrderHeader": {
      "type": "object",
      <!-- Content model of the work order header goes here. -->
    },
    "workOrder": {
      "type": "object",
            <!-- Content model of the work order goes here. -->
    }
  }
}
```

*Example 4: JSON schema illustrating the case when multiple BIEs are selected for expression generation and the schema package option is selected*

## 7.7 Forgoing the Type Inheritance and Attribute Designation.

Type inheritances exists in the CC realm are forgone in the profile BIE JSON schema (this is also the way the BIE entities are represented in the OAGIS repository database – no inheritance. In addition, there is also no notion of type inheritance, i.e., extension, in JSON schema).

JSON property also does not differentiate between meta-property or attribute and regular property of an object like in XML schema (`xsd:attribute` and `xsd:element`). Since the ability to roundtrip between JSON and XML is out of scope, this specification does not specify a special character or character pattern to designate a property as an attribute. The benefits of this design are avoiding potential special character conflict with processors and strange/confusing property name, e.g., `“attribute-typeCode”`, `“attribute-attribute”`.

Examples below illustrate these for the cases of forgoing the complex content extension, simple content restriction, simple type restriction, and attribute designation.

**Complex content type extension example**

Below is a snippet of the model specification of the `Party` element (ASCCP). The associated `PartyType` complex type (ACC) has two inheritances, i.e., it extends the `PartyBaseType` ACC, which in turn extends the `PartyIdentificationType` ACC in the model specification.

```
<xsd:complexType name="PartyIdentificationType" id="oagis-id-00012d42299841139762407l3ed38906">
  <xsd:sequence>
    <xsd:element ref="ID" id="oagis-id-00022d42299841139762407l3ed38906" minOccurs="0" maxOccurs="unbounded" />
    <xsd:element ref="PartyIDSet" id="oagis-id-00032d42299841139762407l3ed38906" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="typeCode" id="oagis-id-00042d42299841139762407l3ed38906" type="CodeType_1E7368"
use="optional"/>
```

```
  <xsd:attribute name="role" id="oagis-id-00052d4229984113976240713ed38906"  type="PartyRoleCodeContentType"
use="optional"/>
</xsd:complexType>
<xsd:complexType name="PartyBaseType" id="oagis-id-00062d4229984113976240713ed38906" abstract="true">
  <xsd:complexContent>
    <xsd:extension base="PartyIdentificationType">
      <xsd:sequence minOccurs="1" maxOccurs="1">
        <xsd:element ref="AccountID" id="oagis-id-00072d4229984113976240713ed38906" minOccurs="0"
maxOccurs="unbounded"/>
        <xsd:element ref="Name" id="oagis-id-00082d4229984113976240713ed38906" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="PartyType" id="oagis-id-00092d4229984113976240713ed38906" abstract="false">
  <xsd:complexContent>
    <xsd:extension base="PartyBaseType">
      <xsd:sequence minOccurs="1" maxOccurs="1">
        <xsd:element name="Extension" id="oagis-id-00102d4229984113976240713ed38906" type="PartyExtensionType"
minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="Party" id="oagis-id-00112d4229984113976240713ed38906" type="PartyType">
```

Example 5 below shows how the `Party` element (ASBIEP) shall appear inside a BIE JSON schema (assuming the `Party` element is a child under the `PurchaseOrderHeader` element (ASBIEP) and its min and max cardinalities are one). Notice that there is no type extension under the "Party" JSON expression, the content of the two based types in the model are merged into the `Party`'s object, i.e., all children of the two based types in the model are direct children properties of the "Party" object.

Notice also that the `typeCode` and `role` attributes in the model (in this case they are BBIE) are expressed as JSON Schema properties just like any other BBIE or ASBIE.

```
{
        "$schema": "http://json-schema.org/draft-04/schema#",
        "required": ["purchaseOrderHeader"],
        "additionalProperties": false,
        "properties": {
                "purchaseOrderHeader": {
                        "description": "Context definition of the purchaseOrderHeader top-level ABIE.",
                        "type": "object",
                        "required": ["Party"],
                        "additionalProperties": false,
                        "properties": {
                                "party": {
                                        "description": "Context definition of purchaseOrderHeader's party ASBIE.",
                                        "type": "object",
                                  "additionalProperties": false,
                                        "properties": {
                                              “typeCode”: {
                                                    // specification of the typeCode attribute.
                                              },
                                              “role” {
                                                    // specification of the role attribute.
                                                    "identifier": {
                                                          // specification of the identifier.
                                                     },
                                                    "partyIdentifierSet": {
                                                          // specification of the party identifier set.
                                                    },
                                                    "accountIdentifier": {
                                                          // specification of the account identifier set.
                                                    },
                                                    "name": {
                                                          // specification of the name
                                                    },
                                                          "extension": {
                                                           // specification of the extention.
                                        }
                                              }

                                        }
                                  }
                            }
                      }
                }
}
```

*Example 5: Forgoing attributing distinction and complex content inheritance hierarchy*

**Simple content type extension example**

In the example below the Name element (BCCP) is bound to `OpenNameType` (BDT). `OpenNameType` is defined in the OAGIS Model with two inheritances through `xsd:extension` based on `NameType` BDT and `NameType_02FC2Z` (BDT) as follows.

```
<xsd:complexType name="OpenNameType" id="oagis-id-3cd82d4229984113976240713ed38906">
      <xsd:simpleContent>
            <xsd:extension base="NameType">
                  <xsd:attribute name="typeCode" type="xsd:token" use="optional" id="oagis-id-
82d03758dfd844bea1676759edf0d653"/>
            </xsd:extension>
      </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="NameType" id="oagis-id-bf66e0afea2c4c2da7bc69af14ca23c9">
      <xsd:simpleContent>
            <xsd:extension base="NameType_02FC2Z">
                  <xsd:attribute name="sequenceNumber" type="xsd:integer" id="oagis-id-
84fa20db74b942449e1885cff79b24df">
                  </xsd:attribute>
```

```
				</xsd:extension>
		</xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="NameType_02FC2Z" id="oagis-id-8ef2aeaecfa645088c4bf4b424905596">
		<xsd:simpleContent>
				<xsd:extension base="xsd:string">
						<xsd:attribute name="languageCode" type="clm56392A20081107_LanguageCodeContentType"
use="optional" id="oagis-id-42e59d799de147b8ab49c8a27ec85ff1">
						</xsd:attribute>
				</xsd:extension>
		</xsd:simpleContent>
</xsd:complexType>
<xsd:element name="Name" type="OpenNameType">
```

The Name element (BBIEP) shall be expressed in the BIE JSON schema fragment as shown in Example 6.

```
{
		“name”: {
				“type”: “object”,
				“required”: [“content”],
				“addtionalProperties”: false,
				“properties”: {
						“content”: {
								“$ref”: “#/definitions/string”,
						},
						“typeCode”: {
								“$ref”: “#/definitions/token”,
						},
						“sequenceNumber”: {
								“$ref”: “#/definitions/integer”
						}
						“languageCode”: {
								“$ref”: “#/definitions/clm56392A20081107_LanguageCode”
						}
				}
		}
		"definitions": {
				"string": {
						"type":"string",
				},
				"token": {
						"type":"string",
				},
				" clm56392A20081107_LanguageCode": {
						"type":"string",
						"enum": ["US-EN", "TH"]
				},
				"integer": {
						"type": "number",
						“multipleOf”: 1
				}
		}
}
```

*Example 6: Forgoing simple content inheritance hierarchy*

Notice the followings:

1. The "content" property is generated for capturing the instance value of all BDTs (see more detail about BDT expression in section 8.5), which is also the instance value of the BBIEP.
3. The type hierarchy from `OpenNameType` to `NameType_02FC2Z` was reduced. All attributes from the two based types become direct properties of the "name" object. Therefore, the "content" property is typed string.
4. In the profile BIE expression, code lists are always defined using a global JSON schema definitions (there is no name clashing because Score ensures that combination of the code list's agency ID, list ID, and version ID are unique. See section 8.7 for complete detail to how the code list type shall be generated).
5. The example also illustrates how an integer type is represented in JSON schema. See section 8.6 for how XML schema-based primitive types are mapped to JSON schema type.

**Simple type restriction example**

In the example below, the `CreationDateTime` element (BCCP) uses `DateTimeType` (BDT) in the model XML schema. `DateTimeType` has two inheritances through a restriction on `DateTimeType_AD9DD9` (BDT), which is in turn a union of a number of types, all of which are based on `xsd:token`.

```
<xsd:element name="CreationDateTime" type="DateTimeType" id="oagis-id-4ba8e6b8c9fb46cda2724a1770fa9baf">
</xsd:element>
<xsd:simpleType name="DateTimeType" id="oagis-id-dd0c8f86b160428da3a82d2866a5b48d">
        <xsd:restriction base="DateTimeType_AD9DD9"/>
</xsd:simpleType>
<xsd:simpleType name="DateTimeType_AD9DD9" id="oagis-id-a5cfd20385314a63afc1ffcf6357a08b" final="union">
        <xsd:union memberTypes=" xbt_CenturyType xbt_DateType xbt_DayOfWeekType …… xbt_YearWeekDayTimeType
xbt_YearWeekDayTimeUTCType xbt_YearWeekDayTimeUTCOffsetType">
        </xsd:union>
</xsd:simpleType>
<xsd:simpleType name="xbt_CenturyType" id="oagis-id-433b017552a14828a92821fd2540d790">
        <xsd:restriction base="xsd:token">
                <xsd:pattern value="[0-9]{2}"/>
        </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="xbt_YearWeekDayTimeUTCOffsetType" id="oagis-id-bd2e6cf3f01a4a90b9ec141fc908f531">
        <xsd:restriction base="xsd:token">
                <xsd:pattern value="[0-9]{4}-W(0[1-9]|[1-4][0-9]|5[0123])-[1-7]T((([01][0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9](|(\.[0-
9]+)))|(24:00:00))([\+|\-]([0-1][0-9]|2[0-3]):[0-5][0-9])"/>
        </xsd:restriction>
</xsd:simpleType>
```

The element `CreationDateTime` (BBIEP) would appear in the BIE JSON schema as shown in Example 7, assuming that the user constrained `CreationDateTime` to `xbt_YearWeekDayTimeUTCOffsetType` in the Score tool.

```
{
        "creationDateTime": {
```

```
			"$ref": "/definitions/xbt_YearWeekDayTimeUTCOffsetType",
		},
		"definitions": {
			"xbt_YearWeekDayTimeUTCOffsetType": {
				"type":"string"
			}
		}
}
```

*Example 7: Forgoing simple type restriction hierarchy*

**Notice the followings:**

1. Because CreationDateTime has no attribute, it is not serialized as a JSON object with the "content" property like the case of Name in Example 6. See more about how a BDT is serialized in section 8.5.
6. xbt_YearWeekDayTimeUTCOffsetType is defined as xsd:token. But xsd:token is eventually mapped to JSON string type, its definition immediately has the string type. See section 8.6 for the type mapping.

## 7.8 Simplified BDT

The BDT serialization is designed to be simple when there is no supplementary component (SC) enabled in the BIE. Taking the Name BDT in Example 6 for instance, if the BIE that uses the Name BDT did not enable any of its SCs including "typeCode", "sequenceNumber", and "languageCode", its serialization would not be a JSON Object but a simple property as shown in Example 8 below.

```
{
		"name": {
			"$ref": "#/definitions/string",
		},
		"definitions": {
			"string": {
				"type":"string",
			}
		}
}
```

*Example 8: An example serialization of a BDT with no SC enabled*

## 7.9 Forgoing the xsd:group

An `xsd:group` in the model specification is imported as an ACC and an ASCCP. However, when a BIE which uses the `xsd:group` is created, there is no ABIE nor ASBIEP created which is corresponding to the ACC and ASCCP `xsd:group`. Instead, associations are established directly between the parent of the group and children of the group. The snippet below shows `FreeFormTextGroup` example in the model XML Schema representation.

```
<xsd:element name="ProductionOrderHeader" type="ProductionOrderHeaderType" id="oagis-id-
111596b8c9fb46cda2724a1770fa1115"/>
<xsd:complexType name="ProductionOrderHeaderType" id="oagis-id-111696b8c9fb46cda2724a1770fa1116">
		<xsd:sequence>
			<xsd:element ref="ID" id="oagis-id-111896b8c9fb46cda2724a1770fa1118" minOccurs="0" maxOccurs="1"/>
			<!-- some other elements here, not shown -->
			<xsd:group ref="FreeFormTextGroup" id="oagis-id-111696b8c9fb46cda2724a1770fa1116"/>
```

```
                <!-- some other elements here, not shown -->
        </xsd:sequence>
</xsd:complexType>
<xsd:group name="FreeFormTextGroup" id="oagis-id-111796b8c9fb46cda2724a1770fa1117">
        <xsd:sequence>
                <xsd:element ref="Description" id="oagis-id-111896b8c9fb46cda2724a1770fa1118" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element ref="Note" id="oagis-id-111
1996b8c9fb46cda2724a1770fa1119" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
</xsd:group>
```

The element `ProductionOrderHeader` (ASBIEP) would appear in the BIE JSON schema fragment as shown in Example 9. Notice that there is no free form text group in the fragment.

```
{
        "productionOrderHeader": {
                "description": "production order header description",
                "type": "object",
                "additionalProperties": false,
                "properties": {
                        “identifier”: {
                                ……
                        }
                        "anotherChildrenOfProductionOrderHeader": {
                                …….
                        },
                        “description”: {
                                “description”: “content model of description in the free form text group would continue
below.”,
                                ………..
                        },
                        “note”: {
                                “description”: “content model of note in the free form text group would continue below.”,
                                ………..
                        }
                }
        }
}
```

*Example 9: Forgoing the group construct*

## 7.10 The “additionalProperties” Key

The “additionalProperties” key shall be present with the value `true`, in all JSON object definition with one exception. The value shall be `true` only when there is an explicit open content in the model such as when `xsd:any` is used in the extension. The `xsd:any` XML Schema construct is represented as “Any Property” ASCCP (and hence ASBIEP) in the repository. Rule 6 formalizes this serialization behavior.

*Rule 6: The “additionalProperties” key with a value `false` must be specified for all JSON schema object definition except when “Any Property” ASCCP or ASBIEP is encountered within the extension element, then the value must be `true`.*

### 7.11 JSON Array

JSON array shall be used when max cardinality of an association (i.e., ASBIE, BBIE, and BBIE SC) is more than 1 or unbounded. Rule 7 below sums up the overall design of the JSON array use.

*Rule 7: JSON array shall be used when max cardinality of an association (i.e., ASBIE, BBIE, and BBIE SC) is more than 1 or unbounded. Its "items" key shall be a JSON schema reflecting the content model of association as described in relevant subsections in section 8. The array shall not declare "additionalItems" key. And its "minItems" and "maxItems" key reflects the min and max cardinalities of the associations, i.e., if min cardinality is zero; "minItems" is not declared; if min cardinality is 1 or more, "minItems" carries the same value; if max cardinality is 0, the association shall not be serialized; if max cardinality is 1 or more, "maxItems" carries the same value; if max cardinality is unbounded, "maxItems" is not declared.*

# 8 Normative Mapping of OAGIS Repository Entities to JSON Schema Constructs

This section formally specifies the maps from table entities (which mostly correspond to CCS entities) in the OAGIS Repository to JSON schema constructs (see the OAGIS Repository Data Model for reference). It indicates how the BIE JSON schema shall be generated. This section includes only the map related to data structure and semantic documentation. Meta-data and context information are out of scope in this version.

In this section, terms with all-cap and under-bar are typically referred to database table or column within the Score tool.

When serializing a BIE, the user selects one or more top-level ASBIEPs to serialized. There are the following options the user can configure the serialization.

1. The user has the option to include all top-level ASBIEPs in the same schema, namely the *schema package* option, or to put each of them in individual schemas.
2. The user has the option to serialize each of the selected top-level ASBIEP as an array.
3. The user has the option to include BIE Documentation in the schema. Throughout the subsections, unless otherwise specifically noted, the "description" keyword is generated only when the user selects the option to include "BIE Documentation".

## 8.1 Top-level ASBIEP

Top-level ASBIEP is serialized as the root schema. The root schema shall have the following JSON Schema keywords populated as follows:

1. The "required" keyword: If the schema package option is selected and there is only one top-level ASBIEP included or if the schema package option is not selected (serialize one top-level BIE per schema), the root schema shall have the JSON Schema "required" keyword indicating the JSON property corresponding to the top-level ASBIEP is mandatory. Conversely, if the schema package option is selected and there are multiple top-level ASBIEPs selected, no "required" keyword shall be present in the root schema. Name of the JSON property corresponding to the top-level ASBIEP shall follow Rule 4 using the top-level ASBIEP's property term.
2. The "additionalProperties" keyword with the value false.
3. The "properties" keyword that is an object containing a subschema or subschemas representing the top-level ABIE or top-level ABIEs used by the corresponding top-level ASBIEP or top-level ASBIEPs.

4. The "definitions" keyword that is an object containing subschemas of primitive types, code lists, and agency identification lists used across the top-level ASBIEPs.

**Lemma 1**: An ASBIEP is derived from ASCCP; hence ASBIEP's property term is the same as ASCCP's property term. An ASCCP is imported from a global or local element declaration that has a complex content in the OAGIS Model XML schema (note that only `DataArea` is a local element in the OAGIS Model XML schema). An ASBIEP uses an ABIE.

## 8.2 Top-level ABIE

Top-level ABIE is an ABIE used by the top-level ASBIEP. A top-level ABIE is serialized as a subschema specifying the content model of the ASBIEP using it as defined in clause 8.1 #3. Name of the subschema shall be generated from the top-level ASBIEP's property term following Rule 4. The subschema shall include the following JSON Schema keywords:

1. The "type" keyword:
   - 1.1. With the value "array", when the top-level ASBIEP is requested to be generated as an array;
   - 1.2. Otherwise, with the value "object".
2. The "description" keyword with the value that is the top-level ABIE's (context) DEFINITION.
3. If the "type" keyword is "object":
   - 3.1. The "required" keyword with the value that is an array of the top-level ABIE's children ASBIEs and BBIEs whose min cardinality value is one or more. The array contains names of JSON properties serialized according to the ASBIEPs and BBIEPs, respectively, used by those ASBIEs and BBIEs. Names of ASBIEPs and BBIEPs are serialized according to Rule 4 using their property terms.
   - 3.2. The "additionalProperties" keyword with the value false.
   - 3.3. The "properties" keyword that is an object containing subschemas of its ASBIE (8.3) and BBIE (8.4) children. If the ASBIE's and BBIE's USED flag is false or MAX_CARDINALITY is zero, they are not serialized.
4. If the "type" keyword is "array":
   - 4.1. The "items" keyword that is an object containing the following keywords:
     - 4.1.1. The "type" keyword with the value "object".
     - 4.1.2. The "required" keyword with the value that is an array of the top-level ABIE's ASBIEs and BBIEs children whose min cardinality value is one or more. The array contains names of JSON properties serialized according the ASBIEPs and BBIEPs, respectively, used by those ASBIEs and BBIEs. Names of the ASBIEPs and BBIEPs are serialized according to Rule 4 using their property terms.
     - 4.1.3. The "additonalProperties" keyword with the value false.
     - 4.1.4. The "properties" keyword that is an object containing subschemas of the top-level ABIE' ASBIE (8.3) and BBIE (8.4) children. However, if an ASBIE's or BBIE's USED flag is false or MAX_CARDINALITY is zero, it is not serialized. Name of each subschema shall be generated from the property term of the ASBIEP or BBIEP used by the respective ASBIE or BBIE children following Rule 4.

**Lemma 2**: An ABIE is derived from an ACC. An ACC is imported from a global complex type declaration with complex content in the OAGIS Model XML schema. An ABIE consists of a number of children associations, each of which, can be an ASBIE or a BBIE.

## 8.3 ASBIE and Its ASBIEP and ABIE

An ASBIEP that is used by an ASBIE is a non-top-level ASBIEP. The non-top-level ASBIEP, in turn, uses an ABIE (in this case the ABIE is a non-top-level ABIE). The ABIE again consists of children associations (ASBIEs and BBIEs) as described in Lemma 2.

For example, if a top-level ASBIEP is a BOM, there is also a BOM top-level ABIE and a child non-top-level ASBIEP can be BOM Header. That means there is an ASBIE that is an association from BOM ABIE to the BOM Header ASBIEP. The BOM Header non-top-level ASBIEP uses a BOM Header non-top-level ABIE (note that ASBIEP and ABIE happen to have the same name, BOM Header, but they are different entities). The BOM Header ABIE again consists of children ASBIEs that are associations to ASBIEPs such as Status, Location and children BBIEs that are associations to BBIEPs such as Identifier and Document Date Time. Such relations repeat down the hierarchy.

The information from the ASBIE and its non-top-level ASBIEP and ABIE is serialized as a property subschema within the "properties" keyword of its parent ABIE as defined in clause 8.2 #0 and #0 and clause 8.3 #0 and #0. Name of the property (i.e., name of the subschema) shall follow Rule 4 using the property term from the respective ASBIEP. The subschema shall have the following keywords:

1. The "type" keyword:
   1.1. If max cardinality of the ASBIE is more than 1 or unbounded:
      1.1.1. The value is "array", if the ASBIE is not `nillable`.
      1.1.2. The value is ["array", null], if the ASBIE is `nillable`.
   1.2. Otherwise:
      1.2.1. The value is "object", if the ASBIE is not `nillable`.
      1.2.2. The value is ["object", null], if the ASBIE is `nillable`.
2. The "description" keyword with the value that is the ASBIE's (context) Definition.
3. If the "type" keyword contains "object":
   3.1. The "required" keyword with the value that is an array of the ABIE's children ASBIEs and BBIEs whose min cardinality value is one or more. The array contains names of JSON properties, serialized according to the ASBIEPs and BBIEPs, respectively, used by those children ASBIEs and BBIEs. Names of the ASBIEPs and BBIEPs are serialized according to Rule 4 using their property terms.
   3.2. The "additonalProperties" keyword with the value false; except when the ASBIEP is `Any Property`, the value shall be `true` and stop here.
   3.3. The "properties" keyword that is an object containing subschemas of the ABIE's ASBIEs (8.3) and BBIEs (8.4) children. If the ASBIE's and BBIE's USED flag is false or MAX_CARDINALITY is zero, they are not serialized.
4. If the "type" keyword contains "array":
   4.1. The "minItems" and "maxItems" keys with the values following Rule 7.
   4.2. The "items" keyword that is a subschema containing the following keywords:
      4.2.1. The "type" keyword with the value "object".
      4.2.2. The "required" keyword with the value that is an array of the ABIE's ASBIEs and BBIEs children whose min cardinality value is one or more. The array contains names of JSON properties serialized according to the ASBIEPs and BBIEPs, respectively, used by those ASBIEs and BBIEs. Names of ASBIEP and BBIEP are serialized according to Rule 4 using their property terms.
      4.2.3. The "additonalProperties" keyword with the value false; except when the ASBIEP is `Any Property`, the value shall be `true` and stop here.
      4.2.4. The "properties" keyword that is an object containing subschemas of the ABIE's children ASBIE (8.3) and BBIE (8.4). If the ASBIE's and BBIE's USED flag is false or MAX_CARDINALITY is zero, they are not serialized.

**Lemma 3:** An ASBIE is derived from an ASCC. An ASCC is an association from an ACC to an ASCCP. An ASCC is imported from an XML Schema element reference or local element within an ACC in the OAGIS Model XML schema.

**Corollary 3.1**: An ASBIE is an association from an ABIE to an ASBIEP. The ASBIEP in turn use another ABIE with or without a qualification. The qualification is the ASBIEP's property term. However, if there is no qualification, the ASBIEP property term is same as the ABIE object class term.

## 8.4 BBIE and Its BBIEP

A BBIE and the BBIEP it associates to are together serialized as a property subschema within the "properties" keyword of its parent ABIE as defined in clause 8.2 #3.3 and #4.1.4 and clause 8.3 #3.3 and #4.2.4. Name of the property (i.e., name of the subschema) shall follow Rule 4 using the BBIEP's property term. The subschema shall have the following keywords:

1. If max cardinality of the BBIE is more than 1 or unbounded,
    1.1. The "type" keyword with the value "array", if the BBIE is not `nillable`.
    1.2. The "type" keyword with the value ["array", null], if the BBIE is `nillable`.
2. Otherwise:
    2.1. When the BBIE has ***no used*** BBIE_SC,
        2.1.1. the "$ref" keyword with the value pointing to a JSON definition subschema according to one of the BBIE's primitive columns[2] that is not null. JSON definition subschemas are the serialization of all primitives (8.6), code lists (8.7), or agency. identification lists (8.7) used within the serialized top-level ABIEs.
        2.1.2. If the BBIE has a default value specified, a keyword "default" with the value according to the BBIE's default value.
        2.1.3. If the BBIE has a fixed value specified, a keyboard "const" with the value according to the BBIE's fixed value.
    2.2. Otherwise (i.e., when the BBIE has at least one BBIE_SC), the "type" keyword with the value "object".
3. The "description" keyword with the value that is the BBIE's (context) DEFINITION.
4. If the "type" keyword is "object":
    4.1. The "required" keyword with the value that is an array of "content" and names of BBIE_SCs whose min cardinality value are one. Names of BBIE_SC are serialized according to Rule 4. The serialized name shall be the truncated concatenation of the BBIE_SC's property term and representation term. However, when the representation term is 'Text', it is dropped.
    4.2. The "additonalProperties" keyword with the value false.
    4.3. The "properties" keyword that is an object containing the following subschemas:
        4.3.1. "content" subschema that is a reference ($ref) to a JSON definition subschema according to one of the BBIE's primitive columns that is not null. JSON definition subschemas are the serialization of all primitives (8.6), code lists (8.7), or agency identification lists (8.7) used within the serialized top-level ABIEs. In addition, the content subschema shall contain "default" and "const" keywords with the value according to the BBIE's default value and fixed value, respectively, if they are specified.

[2] BBIE's primitive columns include BDT_PRI_RESTRI_ID, CODE_LIST_ID, AGENCY_ID_LIST_ID columns.

4.3.2. Subschemas with the names corresponding to the ***used*** BBIE_SCs (section 8.5) whose max cardinality is not zero (used BBIE_SCs are those BBIE_SC turned on by the user).

5. If the "type" keyword contains "array", the "minItems" and "maxItems" keys with the values following Rule 7 and the "items" keyword that is a subschema containing the following keywords:
   5.1. If there is at least one BBIE_SC specified as used and has max cardinality that is not zero,
      5.1.1. The "type" keyword with the value "object".
      5.1.2. The "required" keyword with the value that is an array of "content" and names of BBIE_SCs whose min cardinality values are one. Names of BBIE_SC are serialized according to Rule 4. The serialized name shall be the truncated concatenation of the BBIE_SC's property term and representation term. However, when the representation term is 'Text', it is dropped.
      5.1.3. The "additionalProperties" keyword with the value `false`.
      5.1.4. The "properties" keyword that is an object containing the following subschemas:
         5.1.4.1. "content" subschema that is a reference to a JSON definition according to one of the BBIE's primitive columns that is not null. JSON definition subschemas are the serialization of all primitives (8.6), code lists (8.7), or agency identification lists (8.7) used within the serialized top-level ABIEs. In addition, the content subschema shall contain "default" and "const" keywords with the value according to the BBIE's default value and fixed value, respectively, if they are specified[3].
         5.1.4.2. Subschemas with the names corresponding to the ***used*** BBIE_SCs whose max cardinality is not zero (used BBIE_SCs are those BBIE_SC turned on by the user).
   5.2. Otherwise (i.e., there is no BBIE_SC to serialized), a reference ($ref) to a JSON definition subschema according to one of the BBIE's primitive columns that is not null. JSON definition subschemas are the serialization of all primitives (8.6), code lists (8.7), or agency identification lists (8.7) used within the serialized top-level ABIEs. In addition, the content subschema shall contain "default" and "const" keywords with the value according to the BBIE's default value and fixed value, respectively, if they are specified.

**Lemma 4**: A BBIE is derived from a BCC. A BBIEP is derived from a BCCP. A BCC is an association from an ACC to a BCCP; in other words, a BCC uses a BCCP and hence a BBIE uses a BBIEP. A BCCP is imported from an XML Schema global element declaration that has simple content in the OAGIS Model XML schema or from an XML Schema attribute, which is not characterized as a SC (in other words, when the attribute is a child of an ACC). A BCC is imported from an XML Schema element reference to a BCCP within an ACC.

**Corollary 4.1**: A BBIE is an association from an ABIE to a BBIEP.

**Lemma 5**: A BCCP uses a BDT.

**Corollary 5.1**: A BBIEP also uses a BDT.

**Corollary 5.2**: A BDT may have some SCs, so in the CC realm the relationship chain to an SC looks like BCC -> BCCP -> BDT -> SC. Strictly following the model in the CCS the

[3] The default and fixed value constraint may not make sense in the case that the BBIE is an array. At this time, the application does not provide a warning when that is the case.

relationship chain in the BIE realm would look like BBIE -> BBIEP -> BDT -> SC. However, the OAGIS Repository data model has simplified this by moving both the BDT and SC to have direct relationships to BBIE. That is why the BBIE table has primitive columns including BDT_PRI_RESTRI_ID, CODE_LIST_ID, and AGENCY_ID_LIST_ID; there is a BBIE_SC table that directly links to the BBIE table.

## 8.5 BDT (DT table) and BBIE_SC

Because of Corollary 5.2, a BDT is implicitly generated as part of the BBIE and BBIEP according to clause 8.4 depending on whether there is any SC enabled (used) in the BBIE (as indicated by its associated BBIE_SC record(s)). However, the primitive assigned to the BDT and its SC is generated according to clause 8.6 and 8.7. The primitive can be a built-in type, code list, or agency ID list.

The BBIE_SC is generated as a subschema according to clause 8.4 #4.3.2 and #5.1.4.2. The name of the subschema shall follow Rule 4. The serialized name shall be the truncated concatenation of the BBIE_SC's property term and representation term. However, when the representation term is 'Text', it is dropped. The subschema shall contain the following keywords:

1. The "$ref" keyword whose value refers to the primitive generated in 8.6 or 8.7 according to one of the BBIE_SC's primitive columns[4] that is not null.
2. If the BBIE_SC has a default value specified, a keyword "default" with the value according to the BBIE_SC's default value.
3. If the BBIE has a fixed value specified, a keyboard "const" with the value according to the BBIE_SC's fixed value.

**Lemma 6**: Per CCS, an SC is part of a BDT. However, in the Score and OAGIS Repository implementation the BDTs are CC artifacts and are not reflected explicitly as a BIE artifact. That is, a primitive restriction (which can be a built-in type, code list, or agency ID list) is applied directly in the BBIE and the BBIE_SC tables (as opposed to creating another DT record). For this reason, the primitive used by the BBIE and BBIE_SC should be fetched from the primitive restriction columns captured in the BBIE and BBIE_SC table. The primitive restriction columns in the BBIE_SC table include BDT_SC_PRI_RESTRI_ID, CODE_LIST_ID, AGENCY_ID_LIST_ID columns.

## 8.6 OAGIS Built-in Type (XBT table)

XBT table stores available built-in types including how they map to XML Schema types and JSON Schema types. Only built-in types used in the serialized BIE are generated in the BIE JSON schema. Each built-in type is generated as a subschema within the JSON "definition" subschema. Name of the subschema shall follow the XBT's BUILTIN_TYPE column with the "xsd:" prefix, while the content of the subschema shall include the content from the XBT's JBT_DRAFT05_MAP column. The table in section 10 shows the built-in type map to JSON Schema data type.

[4] BBIE_SC primitive columns include BDT_SC_PRI_RESTRI_ID, CODE_LIST_ID, AGENCY_ID_LIST_ID columns.

### 8.7 Code List and Agency ID List

Code list as well as agency ID list is serialized as a subschema within the JSON "definition" subschema. Only code lists and agency ID lists used in the serialized BIE are generated in the JSON schema. Name of the subschema shall be generated as follows.

The name of the code list shall be the concatenation of the 'cl', AGENCY_ID, '_', VERSION_ID, '_', NAME, 'ContentType', '_', LIST_ID. If NAME is empty, remove the '_' in its front. Also, remove whitespaces in the NAME to make it camel case (LIST_ID and VERSION_ID columns shall not have space).

The name of the agency ID list shall be the concatenation of the 'il', AGENCY_ID_LIST_VALUE, '_', VERSION_ID, '_', NAME, 'ContentType', '_', LIST_ID. Remove whitespaces in the NAME column value, if any, to make it camel case (LIST_ID and VERSION_ID columns shall not have space).

The subschema shall contain the following keywords:

1. The "type" keyword with the value "string".
2. The "enum" keyword which is an array of values of the respective code list or agency ID list.

## 9 Full BIE JSON Schema Examples

This example is for the case where the user has chosen to serialize a single BIE per schema and the BIE Definition enabled.

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "required" : [ "bom" ],
  "additionalProperties" : false,
  "properties" : {
    "bom" : {
      "description" : "This is BOM BIE is for the Super BOM in the context of assemble-to-order.",
      "type" : "object",
      "required" : [ "actionCode", "typeCode" ],
      "additionalProperties" : false,
      "properties" : {
        "typeCode" : {
          "description" : "Indicate that this is a model BOM (i.e., super BOM).",
          "const": "Model",
          "$ref" : "#/definitions/token"
        },
        "actionCode" : {
          "description" : "Indicate the transactional action to perform on the BOM.",
          "$ref" : "#/definitions/cl310_1_oacl_actioncodecontenttype_oagis-id-49c1788460864e99b0872d0a6e58bddb"
        },
        "bomHeader" : {
          "type" : "object",
          "required" : [ "identifier" ],
          "additionalProperties" : false,
          "properties" : {
            "identifier" : {
              "description" : "This is the middleware generate identifier of the BOM used for cross reference.",
              "type" : "object",
              "required" : [ "content", "schemeIdentifier" ],
              "additionalProperties" : false,
              "properties" : {
                "content" : {
                  "$ref" : "#/definitions/normalizedString"
                },
                "schemeIdentifier" : {
                  "$ref" : "#/definitions/token",
                  "const": "xref"
                }
              }
            },
            "documentIdentifierSet" : {
              "type" : "object",
              "required" : [ "identifier" ],
              "additionalProperties" : false,
              "properties" : {
                "identifier" : {
                  "description" : "Internal primary key of the associated entity.",
                  "type" : "object",
                  "required" : [ "content", "schemeIdentifier", "schemeAgencyIdentifier" ],
                  "additionalProperties" : false,
                  "properties" : {
                    "content" : {
                      "$ref" : "#/definitions/normalizedString"
                    },
                    "schemeIdentifier" : {
                      "description" : "The Internal Key indicates that this identifier is the primary key of the associated entity.",
                      "$ref" : "#/definitions/token",
                      "const": "Internal Key"
                    },
                    "schemeAgencyIdentifier" : {
                      "description" : "This should indicate the ID of the source application that owns this identifier.",
                      "$ref" : "#/definitions/token"
                    }
                  }
                }
              }
            },
            "effectiveTimePeriod" : {
              "description" : "Time period in which the BOM is valid.",
              "type" : "object",
              "required" : [ "startDateTime" ],
              "additionalProperties" : false,
              "properties" : {
                "startDateTime" : {
                  "description" : "The date time when the BOM becomes effective. This is required.",
                  "$ref" : "#/definitions/dateTime"
                },
                "endDateTime" : {
                  "description" : "End time is optional. When it is not specified, the effectivity is open ended.",
                  "$ref" : "#/definitions/dateTime"
                }
              }
            }
          }
```

```
      },
      "bomItemData" : {
        "type" : "array",
        "items" : {
          "type" : "object",
          "required" : [ "identifier" ],
          "additionalProperties" : false,
          "properties" : {
            "identifier" : {
              "description" : "This is an identifier generated by the middleware that can be used for cross-reference.",
              "type" : "object",
              "required" : [ "content", "schemeIdentifier" ],
              "additionalProperties" : false,
              "properties" : {
                "content" : {
                  "$ref" : "#/definitions/normalizedString"
                },
                "schemeIdentifier" : {
                  "$ref" : "#/definitions/token",
                  "const": "xref"
                }
              }
            },
            "itemIdentifierSet" : {
              "type" : "object",
              "required" : [ "identifier" ],
              "additionalProperties" : false,
              "properties" : {
                "identifier" : {
                  "description" : "Application internal primary key of the associated entity.",
                  "type" : "object",
                  "required" : [ "content", "schemeIdentifier", "schemeAgencyIdentifier" ],
                  "additionalProperties" : false,
                  "properties" : {
                    "content" : {
                      "$ref" : "#/definitions/normalizedString"
                    },
                    "schemeIdentifier" : {
                      "description" : "The Internal Key indicates that this identifier is the primary key of the associated entity.",
                      "$ref" : "#/definitions/token",
                      "const": "Internal Key"
                    },
                    "schemeAgencyIdentifier" : {
                      "description" : "This should indicate the ID of the source application that owns this identifier.",
                      "$ref" : "#/definitions/token"
                    }
                  }
                }
              }
            },
            "quantity" : {
              "description" : "Quantity of the item used in the BOM.",
              "type" : "array",
              "items" : {
                "type" : "object",
                "required" : [ "content" ],
                "additionalProperties" : false,
                "properties" : {
                  "content" : {
                    "$ref" : "#/definitions/integer"
                  },
                  "typeCode" : {
                    "description" : "Indicate min, max, or fixed quantity. If there is a fixed quantity there shall be no min or max
quantity.",
                    "$ref" : "#/definitions/cl402_1.0_bomquantitytypecodecontenttype_oagis-id-dafdf58b5c8246e098ab640547da91ca"
                  },
                  "unitCode" : {
                    "$ref" : "#/definitions/cl402_1.0_oacl_unitcode_greenmanufacturingextensioncontenttype_oagis-id-
0ae356b7d83a427680330b9662b48c0e"
                  }
                }
              }
            },
            "note" : {
              "description" : "Remark to display to the user when viewing the item details.",
              "type" : "array",
              "items" : {
                "$ref" : "#/definitions/string"
              }
            }
          }
        }
      },
      "bomOption" : {
        "description" : "Specifies BOM Item that is an option and the characteristics of the option.",
        "type" : "array",
        "items" : {
          "type" : "object",
          "required" : [ "identifier" ],
          "additionalProperties" : false,
          "properties" : {
            "identifier" : {
              "description" : "This is an identifier generated by the middleware that can be used for cross-reference.",
              "type" : "object",
              "required" : [ "content" ],
              "additionalProperties" : false,
              "properties" : {
                "content" : {
                  "$ref" : "#/definitions/normalizedString"
                },
                "schemeIdentifier" : {
                  "$ref" : "#/definitions/token",
                  "const": "xref"
                }
              }
            },
            "note" : {
              "description" : "Convey instruction to the user when configuring the option.",
              "type" : "array",
              "items" : {
                "$ref" : "#/definitions/string"
              }
          },
          "bomItemData" : {
            "type" : "array",
            "items" : {
              "type" : "object",
              "additionalProperties" : false,
              "properties" : {
                "identifier" : {
                  "description" : "This is the reference back to the Identifier in the BOM Item Data.",
                  "$ref" : "#/definitions/normalizedString",
                  "const": "xref"
                }
              }
            }
          },
          "quantity" : {
            "description" : "Quantity in the BOM Option can override the Quantity in the BOM Item Data.",
            "type" : "array",
            "items" : {
              "type" : "object",
              "required" : [ "content", "typeCode", "unitCode" ],
              "additionalProperties" : false,
              "properties" : {
                "content" : {
                  "$ref" : "#/definitions/decimal"
                },
                "typeCode" : {
                  "$ref" : "#/definitions/cl402_1.0_bomquantitytypecodecontenttype_oagis-id-dafdf58b5c8246e098ab640547da91ca"
                },
                "unitCode" : {
                  "$ref" : "#/definitions/cl402_1.0_oacl_unitcode_greenmanufacturingextensioncontenttype_oagis-id-
0ae356b7d83a427680330b9662b48c0e"
                }
              }
            }
          }
        }
      }
    }
  }
},
"definitions" : {
  "token" : {
    "type" : "string"
  },
  "cl310_1_oacl_actioncodecontenttype_oagis-id-49c1788460864e99b0872d0a6e58bddb" : {
    "type" : "string",
    "enum" : [ "Add", "Change", "Delete", "Replace", "UpSert", "Accepted", "Modified", "Rejected" ]
  },
  "normalizedString" : {
    "type" : "string"
  },
  "dateTime" : {
    "type" : "string",
    "format" : "date-time"
    },
    "integer" : {
      "type" : "number",
      "multipleOf" : 1
    },
    "cl402_1.0_bomquantitytypecodecontenttype_oagis-id-dafdf58b5c8246e098ab640547da91ca" : {
      "type" : "string",
      "enum" : [ "MN", "MX", "FX" ]
    },
    "cl402_1.0_oacl_unitcode_greenmanufacturingextensioncontenttype_oagis-id-0ae356b7d83a427680330b9662b48c0e" : {
      "type" : "string",
      "enum" : [ "EA", "PR" ]
    },
    "string" : {
      "type" : "string"
    },
    "decimal" : {
      "type" : "number"
    }
  }
}
```

## 10 OAGIS Built-in Primitive Type Mapping to JSON Types

| BUILTIN_TYPE | JBT_DRAFT05_MAP |
|---|---|
| `xsd:anyType` | `{"type":"string"}` |
| `xsd:anySimpleType` | `{"type":"string"}` |
| `xsd:duration` | `{"type":"string", "pattern":"^[-]?P(?!$)(?:\\d+Y)?(?:\\d+M)?(?:\\d+D)?(?:T(?!$)(?:\\d+H)?(?:\\d+M)?(?:\\d+(?:\\.\\d+)?S)?)?$"}` |
| `xsd:dateTime` | `{"type":"string", "format":"date-time"}` |
| `xsd:time` | `{"type":"string"}` |
| `xsd:date` | `{"type":"string"}` |
| `xsd:gYearMonth` | `{"type":"string"}` |
| `xsd:gYear` | `{"type":"string"}` |
| `xsd:gMonthDay` | `{"type":"string"}` |
| `xsd:gDay` | `{"type":"string"}` |
| `xsd:gMonth` | `{"type":"string"}` |
| `xsd:string` | `{"type":"string"}` |
| `xsd:normalizedString` | `{"type":"string"}` |
| `xsd:token` | `{"type":"string"}` |
| `xsd:language` | `{"type":"string"}` |
| `xsd:boolean` | `{"type":"boolean"}` |

| BUILTIN_TYPE | JBT_DRAFT05_MAP |
|---|---|
| xsd:base64Binary | {"type":"string"} |
| xsd:hexBinary | {"type":"string"} |
| xsd:float | {"type":"number"} |
| xsd:decimal | {"type":"number"} |
| xsd:integer | {"type":"number",<br>"multipleOf":1} |
| xsd:nonNegativeInteger | {"type":"integer",<br>"minimum":0,<br>"exclusiveMinimum":false} |
| xsd:positiveInteger | {"type":"integer",<br>"minimum":0,<br>"exclusiveMinimum":true} |
| xsd:double | {"type":"number"} |
| xsd:anyURI | {"type":"string"<br>"format":"uriref"} |
| xbt_BooleanType | {"type":"boolean"} |
| xbt_WeekDurationType | {"type":"string"} |
| xbt_CenturyType | {"type":"string"} |
| xbt_DateType | {"type":"string"} |
| xbt_DayOfWeekType | {"type":"string"} |
| xbt_DayOfYearType | {"type":"string"} |
| xbt_DayType | {"type":"string"} |
| xbt_MonthDayType | {"type":"string"} |
| xbt_MonthType | {"type":"string"} |
| xbt_WeekType | {"type":"string"} |
| xbt_WeekDayType | {"type":"string"} |
| xbt_YearDayType | {"type":"string"} |
| xbt_YearMonthType | {"type":"string"} |
| xbt_YearType | {"type":"string"} |
| xbt_YearWeekType | {"type":"string"} |
| xbt_YearWeekDayType | {"type":"string"} |
| xbt_HourMinuteType | {"type":"string"} |
| xbt_HourMinuteUTCType | {"type":"string"} |
| xbt_HourMinuteUTCOffsetType | {"type":"string"} |
| xbt_HourType | {"type":"string"} |
| xbt_HourUTCType | {"type":"string"} |
| xbt_HourUTCOffsetType | {"type":"string"} |
| xbt_MinuteType | {"type":"string"} |

| BUILTIN_TYPE | JBT_DRAFT05_MAP |
|---|---|
| xbt_MinuteSecondType | {"type":"string"} |
| xbt_SecondType | {"type":"string"} |
| xbt_TimeType | {"type":"string"} |
| xbt_TimeUTCType | {"type":"string"} |
| xbt_TimeUTCOffsetType | {"type":"string"} |
| xbt_DateHourMinuteType | {"type":"string"} |
| xbt_DateHourMinuteUTCType | {"type":"string"} |
| xbt_DateHourMinuteUTCOffsetType | {"type":"string"} |
| xbt_DateHourType | {"type":"string"} |
| xbt_DateHourUTCType | {"type":"string"} |
| xbt_DateHourUTCOffsetType | {"type":"string"} |
| xbt_DateTimeType | {"type":"string"} |
| xbt_DateTimeUTCType | {"type":"string"} |
| xbt_DateTimeUTCOffsetType | {"type":"string"} |
| xbt_DayHourMinuteType | {"type":"string"} |
| xbt_DayHourMinuteUTCType | {"type":"string"} |
| xbt_DayHourMinuteUTCOffsetType | {"type":"string"} |
| xbt_DayHourType | {"type":"string"} |
| xbt_DayHourUTCType | {"type":"string"} |
| xbt_DayHourUTCOffsetType | {"type":"string"} |
| xbt_DayOfWeekHourMinuteType | {"type":"string"} |
| xbt_DayOfWeekHourMinuteUTCType | {"type":"string"} |
| xbt_DayOfWeekHourMinuteUTCOffsetType | {"type":"string"} |
| xbt_DayOfWeekHourType | {"type":"string"} |
| xbt_DayOfWeekHourUTCType | {"type":"string"} |
| xbt_DayOfWeekHourUTCOffsetType | {"type":"string"} |
| xbt_DayOfWeekTimeType | {"type":"string"} |
| xbt_DayOfWeekTimeUTCType | {"type":"string"} |
| xbt_DayOfWeekTimeUTCOffsetType | {"type":"string"} |
| xbt_DayOfYearHourMinuteType | {"type":"string"} |
| xbt_DayOfYearHourMinuteUTCType | {"type":"string"} |
| xbt_DayOfYearHourMinuteUTCOffsetType | {"type":"string"} |
| xbt_DayOfYearHourType | {"type":"string"} |
| xbt_DayOfYearHourUTCType | {"type":"string"} |
| xbt_DayOfYearHourUTCOffsetType | {"type":"string"} |
| xbt_DayOfYearTimeType | {"type":"string"} |
| xbt_DayOfYearTimeUTCType | {"type":"string"} |

| BUILTIN_TYPE | JBT_DRAFT05_MAP |
|---|---|
| xbt_DayOfYearTimeUTCOffsetType | {"type":"string"} |
| xbt_DayTimeType | {"type":"string"} |
| xbt_DayTimeUTCType | {"type":"string"} |
| xbt_DayTimeUTCOffsetType | {"type":"string"} |
| xbt_MonthDayHourMinuteType | {"type":"string"} |
| xbt_MonthDayHourMinuteUTCType | {"type":"string"} |
| xbt_MonthDayHourMinuteUTCOffsetType | {"type":"string"} |
| xbt_MonthDayHourType | {"type":"string"} |
| xbt_MonthDayHourUTCType | {"type":"string"} |
| xbt_MonthDayHourUTCOffsetType | {"type":"string"} |
| xbt_MonthDayTimeType | {"type":"string"} |
| xbt_MonthDayTimeUTCType | {"type":"string"} |
| xbt_MonthDayTimeUTCOffsetType | {"type":"string"} |
| xbt_WeekDayHourMinuteType | {"type":"string"} |
| xbt_WeekDayHourMinuteUTCType | {"type":"string"} |
| xbt_WeekDayHourMinuteUTCOffsetType | {"type":"string"} |
| xbt_WeekDayHourType | {"type":"string"} |
| xbt_WeekDayHourUTCType | {"type":"string"} |
| xbt_WeekDayHourUTCOffsetType | {"type":"string"} |
| xbt_WeekDayTimeType | {"type":"string"} |
| xbt_WeekDayTimeUTCType | {"type":"string"} |
| xbt_WeekDayTimeUTCOffsetType | {"type":"string"} |
| xbt_YearDayHourMinuteType | {"type":"string"} |
| xbt_YearDayHourMinuteUTCType | {"type":"string"} |
| xbt_YearDayHourMinuteUTCOffsetType | {"type":"string"} |
| xbt_YearDayHourType | {"type":"string"} |
| xbt_YearDayHourUTCType | {"type":"string"} |
| xbt_YearDayHourUTCOffsetType | {"type":"string"} |
| xbt_YearDayTimeType | {"type":"string"} |
| xbt_YearDayTimeUTCType | {"type":"string"} |
| xbt_YearDayTimeUTCOffsetType | {"type":"string"} |
| xbt_YearWeekDayHourMinuteType | {"type":"string"} |
| xbt_YearWeekDayHourMinuteUTCType | {"type":"string"} |
| xbt_YearWeekDayHourMinuteUTCOffsetType | {"type":"string"} |
| xbt_YearWeekDayHourType | {"type":"string"} |
| xbt_YearWeekDayHourUTCType | {"type":"string"} |
| xbt_YearWeekDayHourUTCOffsetType | {"type":"string"} |

| BUILTIN_TYPE | JBT_DRAFT05_MAP |
|---|---|
| xbt_YearWeekDayTimeType | {"type":"string"} |
| xbt_YearWeekDayTimeUTCType | {"type":"string"} |
| xbt_YearWeekDayTimeUTCOffsetType | {"type":"string"} |