The Global Language of Business

Semantic Data Modelling Technical Bulletin

Normatively referenced by the GS1 System Architecture Release 10.0, Section 6.5. GS1 data models can be defined independently of data formats, using Semantic Web/Linked Data standards.

Release 1.0, Approved, Jun 2021

1 Introduction

As a Standards Development Organisation, GS1 oversees the community standardisation of data models and interfaces for the efficient exchange of data, whether master data, transaction data or event data.

For future-proofing this data modelling work, it is important that data models and interfaces are defined at an abstract layer, independent of any particular syntax so that as new data formats emerge, the semantic data model does not change at the abstract layer but a new data binding is defined for the new data format. This is consistent with the GS1 Architecture Principles of Technology Independence and Re-Use of Components.

Although XML has long been used for exchange of information via structured documents, alternative data formats such as JSON (JavaScript Object Notation) and JSON-LD (JSON for Linked Data) are rapidly gaining traction as alternatives. JSON is a much simpler data format than XML and expresses lists/arrays as well as associative arrays (key:value lookup tables). JSON is often natively supported by modern programming and scripting languages and is useful for exchanging structured data between systems that might use different programming languages. Many developers find it much easier to work with JSON than with XML. However, JSON lacks some of the features of XML, including support for multiple namespaces. The W3C JSON-LD standard defines a version of JSON that is also a Linked Data format with support for multiple namespaces and explicit data types.

Linked Data provides a lowest common denominator across various data formats and includes standardised tools and techniques for easier data integration across multiple information systems. This capability to query across multiple information systems can be useful when performing an upstream root cause analysis to find the source of a defective product, as well as performing subsequent downstream impact analysis, to identify other products and product instances that might also have been affected by the source of the defect (such as a defective or contaminated ingredient, mis-calibrated or defective processing equipment etc.).

Regardless of whether the data is exchanged in XML, JSON, JSON-LD or any other format or via Web Services, AS2 or Open API / REST Web APIs or other protocols, the meanings (semantics) of the components of our standardised data model should be defined in a way that can be used consistently in any data format and through any interface protocol.

2 Building blocks of a data model

The main building blocks of a data model include:

- Classes
- Properties (also called data fields, attributes, predicates)
- Code Lists and their values

Every component within a data model (whether a class, property, code list or value within a code list) is a term. Each term should have an identifier, a human-friendly name or label, ideally in multiple languages, and a precise semantic definition.

Semantic Web / Linked Data technologies recommend the use of Web URIs as globally unambiguous Web-friendly identifiers for every term within a data model. Standards such as UML and Linked Data standards such as RDF, RDFS, OWL and SKOS provide a way of describing a data model and its structure in an abstract way, independent of syntax, that supports multi-lingual names/labels, definitions and other intrinsic details about meaning and usage.

Models comprising classes, properties and code lists form the basis of any meaningful exchange of data. However, they can be used by different actors in different contexts. A term that might be essential to one kind of transaction may be optional or even irrelevant to another.

The existence of a term does not itself require its use. It is therefore not possible to validate a given dataset against a data model in isolation.

A further building block is required for this: a profile. This is a set of constraints on the usage of terms from one or more data models that define cardinalities (how many times a term MAY or MUST be used), and it might also include further terms or alternative code lists.

2.1 Identifiers and namespaces

Web URIs have the extremely useful characteristic that they can be looked up, usually by simply clicking them. You can 'follow your nose' across the Web to the data you want. Like any global identifier scheme, they have structure. Within that structure, the domain name is crucial as it defines the authority for the rest of the identifier. So, for example, https://www.smh.com.au/politics identifies a different resource than https://edition.cnn.com/politics, even though they both end with 'politics'. The usual term for this authority is namespace.

Useful though they clearly are, Web URIs are clunky and relatively long. If we want to use terms from the GS1 Web vocabulary, for example, it's inconvenient to have to include the whole URI. We do not really want to have to write terms like https://gs1.org/voc/additive and https://gs1.org/voc/bestBeforeDate in full every time, especially as the majority of the identifier is repeated from one to another.

The answer is to define a short prefix that can be substituted for the namespace. For example, we can define 'gs1:' to mean 'https://gs1.org/voc/'. This allows us then to use the shorthand gs1:additive and gs1:bestBeforeDate rather than the full URI. As well as being shorter, it's a lot more readable. These shortened URIs are known as Compact URIs or CURIEs [CURIE].

Since prefixes are just short strings, they are *not* guaranteed to be globally unique. In any machine-readable data, they must be tied to the base URI for which they can be substituted. It would be possible, although very unhelpful and bad practice, to define the term 'gs1:' to replace a different base URL. Thankfully, this is never done in practice. By convention, 'gs1:' always means 'https://gs1.org/voc/', 'schema:' always means 'https://schema.org/' and 'xsd:' always means 'http://www.w3.org/2001/XMLSchema#'. There's a handy look up service for widely-used conventional prefixes at https://prefix.cc.

2.2 Classes

Classes (sometimes called data objects) group collections of related properties together and often correspond to a real-world thing such as a product or asset, invoice or something less tangible, such as a traceability observation or sensor measurement or department within an organisation.

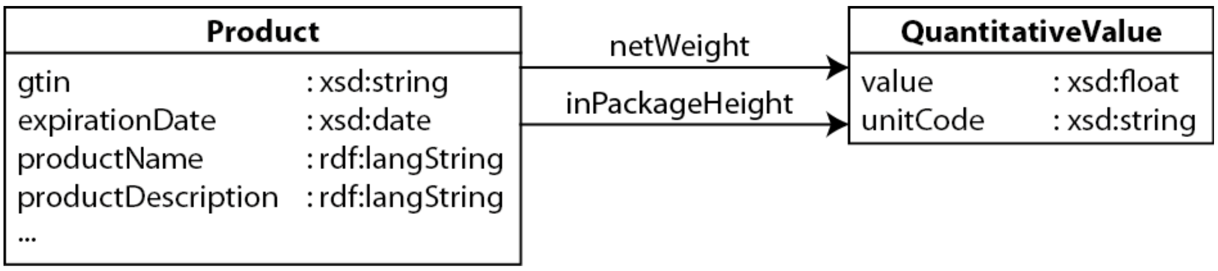


Figure 1 – simple examples of classes, shown in a UML class diagram

Figure 1 shows two simple examples of classes defined in schema.org and the GS1 Web vocabulary. In this example, the `Product` class represents real-world products or trade items and defines a number of properties that are useful for describing a product and distinguishing one product from another. The diagram only shows a small sample of the properties defined for the `Product` class. In contrast, a `QuantitativeValue` class does not correspond to a tangible real-world thing, but to a concept or a useful collection of combination of interdependent properties. The `QuantitativeValue` class (referred to as `Measurement` in the GDSN data model) includes properties such as `value` and `unitCode`. To express a measurement of weight, length, temperature etc. it is necessary to be able to specify not only the floating-point value but also a unit code, such a UN ECE Rec20 code string. The `QuantitativeValue` class acts as a container for the `value` and `unitCode` properties and ensures that if there are multiple measurements (e.g. `netWeight`, `inPackageHeight` etc.), there is no ambiguity about which unit code corresponds to which floating-point value because they are always paired together within a `QuantitativeValue` class.

2.2.1 Sub-Classes

Classes can be organised in a hierarchy, in which subclasses represent a subset of the superclass that has some specialised characteristics. For example, Vegetables, Fruit, Meat, Poultry, Seafood are all subclasses of a superclass, Food. Further subclasses can be defined. For example, Legumes and Root Vegetables are subclasses of Vegetables. Apples, Oranges, Bananas, Melons and Berries are subclasses of Fruit.

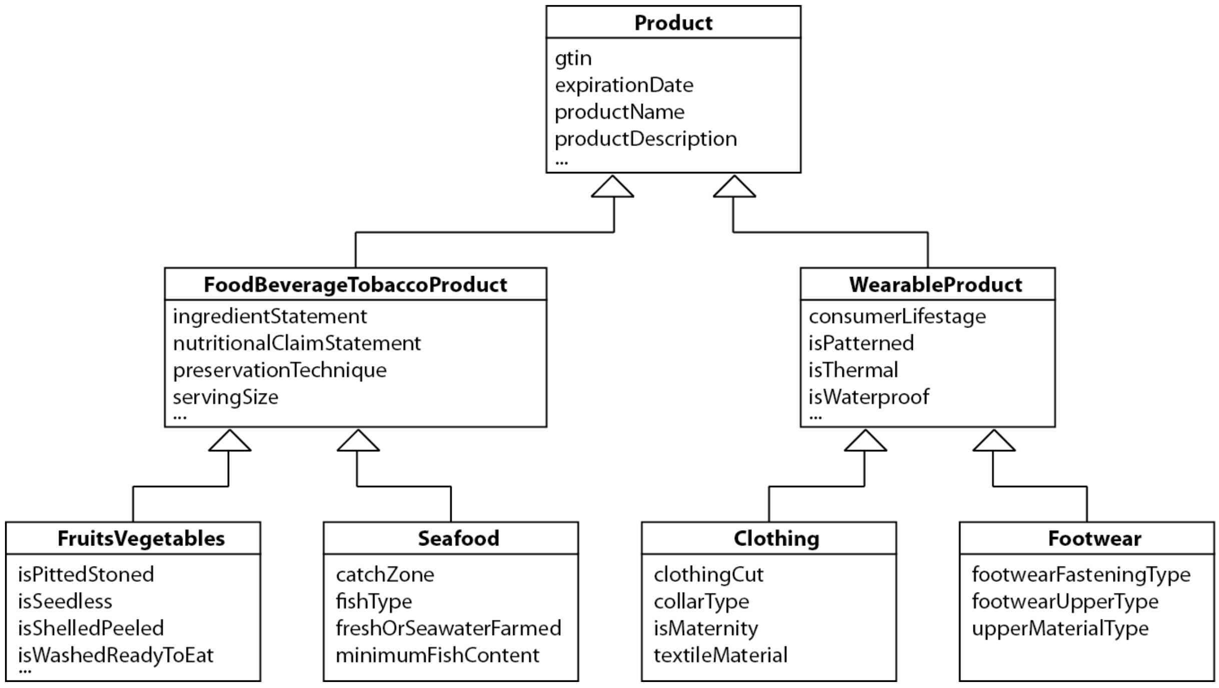


Figure 2 – examples of subclasses and inheritance of properties

Figure 2 shows examples of a hierarchy of classes and subclasses defined within the GS1 Web vocabulary. In this example, the `Product` class is the superclass and `FoodBeverageTobaccoProduct` class and `WearableProduct` class are subclasses of the `Product` class. Each of these subclasses represent specialised subsets of the `Product` superclass and they define properties that are relevant within members of their own subclass. For example, `WearableProduct` defines a property `isWaterproof`. `FoodBeverageTobaccoProduct` defines a property `ingredientStatement`.

Any properties defined for a class are also available to any subclasses below them in the hierarchy, while subclasses may define their own properties in addition to the properties that they inherit from superclasses above them in the hierarchy.

For example, every member of the `Clothing` class can use any of the properties defined within the `Clothing` class (such as `textileMaterial`), as well as any of the properties defined within the `WearableProduct` class (such as `isWaterproof`) as well as any of the properties defined within the `Product` class (such as `productDescription` or `gtin`).

2.3 Properties

Properties (also called data fields, attributes, predicates) express a particular relationship and a value associated with that relationship. As shown in Figure 3, some properties (called 'data type' properties) expect a simple literal value such as a string, language-tagged string, date, integer or floating-point value.

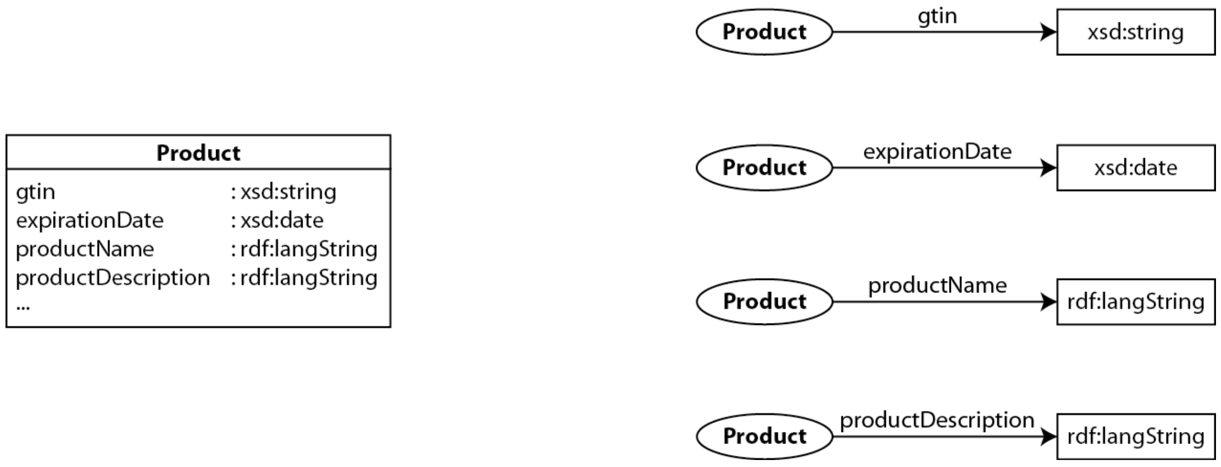


Figure 3 – examples of 'data type' properties that expect a simple literal value

Other properties (such as `minimumFishContent` shown in Figure 4 or `netWeight` shown in Figure 1) expect a class or complex data object (such as `QuantitativeValue`) that itself expresses further properties, such as `value` and `unitCode`.

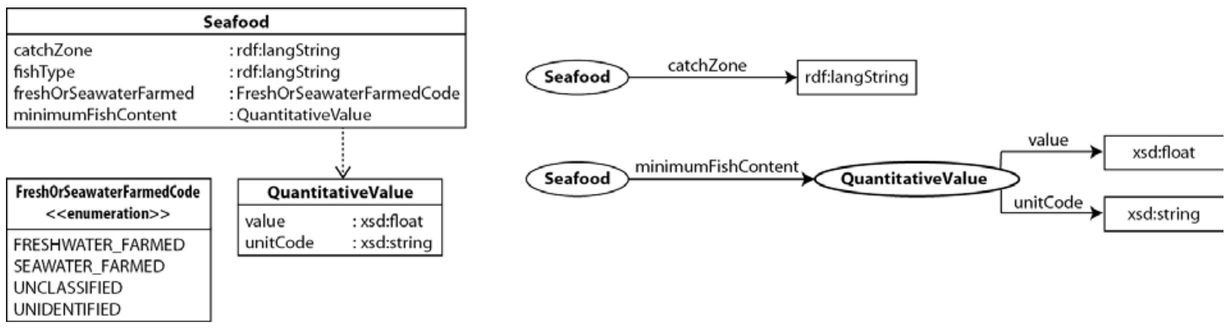


Figure 4 – `minimumFishContent` is an example of an object property whose value is a class or complex data object.

2.3.1 Sub-Properties

Sub-properties are used to take the meaning of a more general property and to make it more specific.

For example, the schema.org vocabulary has a general purpose property https://schema.org/name

The GS1 Web vocabulary defines a number of sub-properties of this, such as https://www.gs1.org/voc/productName (for the names of products) and https://www.gs1.org/voc/organizationName (for the names of organisations).

2.3.2 Domain of a Property (Class in which it is defined)

The domain of a property indicates the class in which it is defined. Figure 5 shows a couple of examples of properties defined within the GS1 Web vocabulary and indicates the domain and range for each.

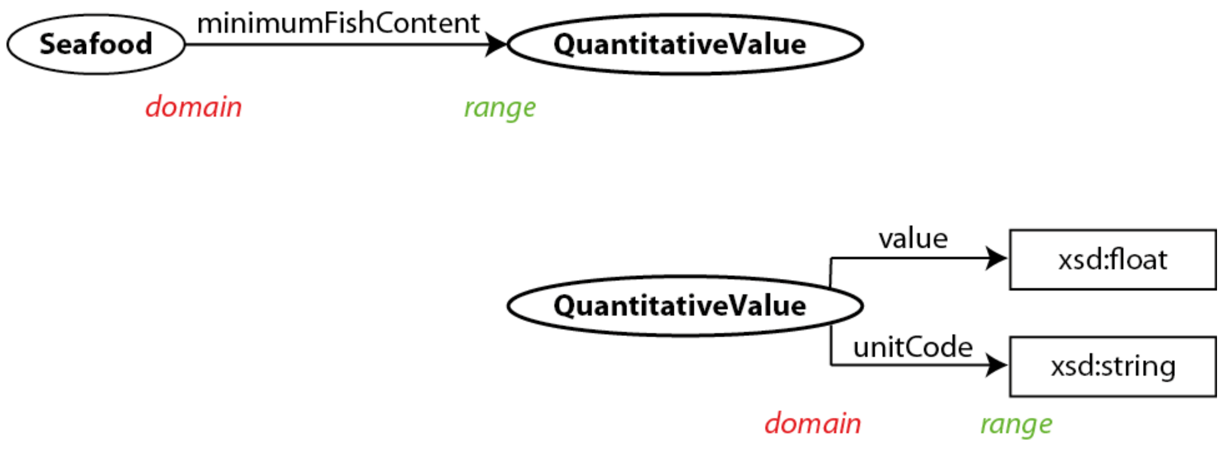


Figure 5 – examples of properties and their domain and range.

In the examples shown in Figure 5, the *domain* of the `minimumFishContent` property is the `Seafood` class, whereas the *domain* of the `unitCode` property is the `QuantitativeValue` class.

2.3.3 Range of a Property (Expected value type)

The range of a property indicates the expected data type for the value of the property. In the examples shown in Figure 5, the *range* of the `minimumFishContent` property is the `QuantitativeValue` class, and the *range* of the `unitCode` property is an `xsd:string` literal value (a string, in this case a UN ECE Rec 20 unit code).

2.4 Code Lists

Code Lists (sometimes called enumerations) are used to define a set of defined code values and meanings / definitions. The main advantages of using code lists are:

- code lists limit the number of choices to a finite set, unlike a free-form text field
- everyone globally uses the same code value instead of a free-form text string that needs to be translated into different human languages
- each code value can be used globally and associated with a human-readable meaning or definition can be translated into various human languages

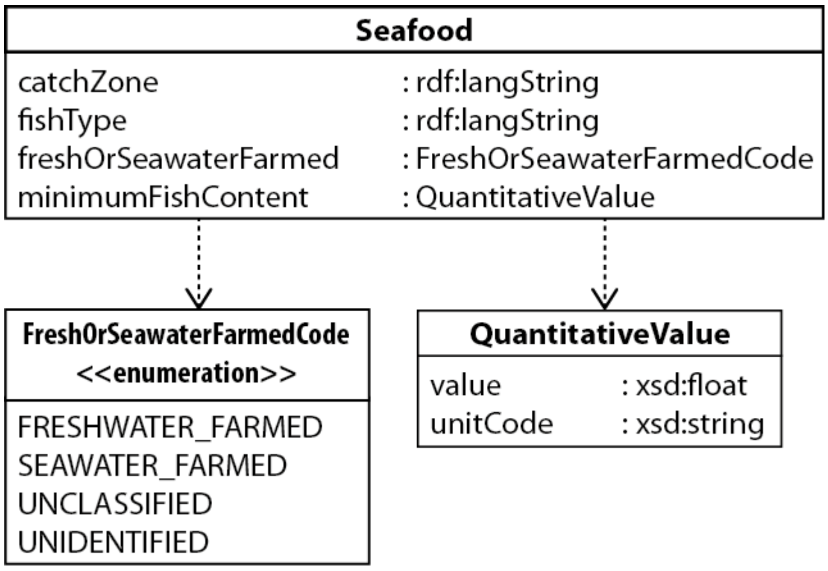

Seafood
catchZone : rdf:langString
fishType : rdf:langString
freshOrSeawaterFarmed : FreshOrSeawaterFarmedCode
minimumFishContent : QuantitativeValue
FreshOrSeawaterFarmedCode
<<enumeration>>
FRESHWATER_FARMED
SEAWATER_FARMED
UNCLASSIFIED
UNIDENTIFIED
QuantitativeValue
value : xsd:float
unitCode : xsd:string

Figure 6 – `FreshOrSeawaterFarmedCode` is an example of a code list or enumeration.

A code list is modelled as a class that does not have any associated properties; enumerated defined values within the code list are members of the class that represents the code list. Figure 6 shows the use of a code list; `FreshOrSeawaterFarmedCode` is a code list (or enumeration within a UML class diagram) and defines some code values to be used on a global basis. Each code value, such as `FRESHWATER_FARMED` is used without translation for interoperable global data exchange, although each code value might have defined labels translated into multiple human languages, such as "freshwater farmed" in English, "élevé en eau douce" in French, "in Süßwasser gezüchtet" in German.

In this way, software that needs to take a decision based upon the standardised global code value does not need to be concerned with translation into various human languages at the data exchange layer, only at the user-interface or presentation layer, when the code value within the code list might be displayed in a Web interface as a pull-down menu or label against a checkbox, showing the label for the code value in the user's preferred language.

3 Use of W3C Linked Data standards for semantic modelling

The World Wide Web Consortium (W3C) has defined a number of technical recommendations (standards) that are useful for expressing a semantic data model at an abstract layer, independent of syntax. Effectively they provide a machine-interpretable equivalent of a UML class diagram that a human can interpret.

The foundation of Linked Data are URIs (primarily Web URIs / URLs) and Resource Description Framework (RDF) in which any arbitrary data structure can be collapsed to a set of RDF **triples** that connect a Subject via a Property to a Value, as shown in Figure 7. A RDF triple can be viewed as a simple logical fact / assertion or 3-word sentence, Subject –> Property –> Value.

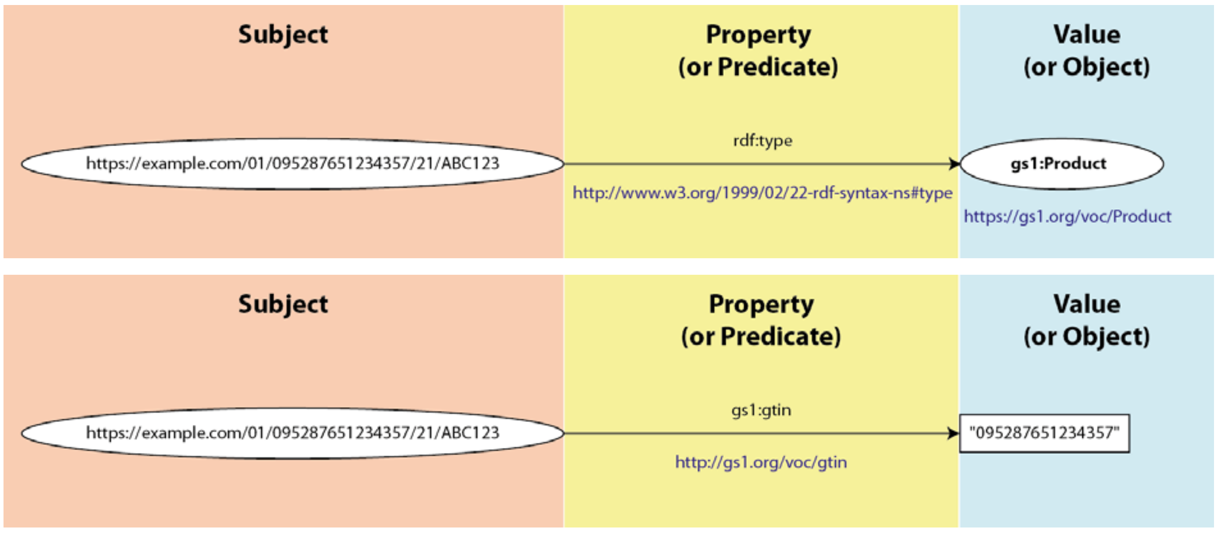

Subject
Property (or Predicate)
Value (or Object)
https://example.com/01/095287651234357/21/ABC123
rdf:type
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
gs1:Product
https://gs1.org/voc/Product
Subject
Property (or Predicate)
Value (or Object)
https://example.com/01/095287651234357/21/ABC123
gs1:gtin
http://gs1.org/voc/gtin
"095287651234357"

Figure 7 – simple examples of RDF triples to express a Subject – Property – Value relationship.

The two RDF triples in Figure 7 assert that:

1. The thing identified by GS1 Digital Link URI https://example.com/01/09528765123457/21/ABC123 is a Product (a member of the gs1:Product class, whose Web URI is https://gs1.org/voc/Product)
   The property rdf:type (= http://www.w3.org/1999/02/22-rdf-syntax-ns#type) links an individual to a class to which it belongs.
2. The thing identified by GS1 Digital Link URI https://example.com/01/09528765123457/21/ABC123 has a GTIN value of "09528765123457".
   The property gs1:gtin (= https://gs1.org/voc/gtin) links a thing to its GTIN identifier.

Note that in Linked Data, every class, property, code list or code value within a code list has a corresponding Web URI, which enables the corresponding name, definition and usage details to be easily retrieved by making a Web request. Linked Data often makes use of Compact URI Expressions [CURIE] to more compactly express a long Web URI as a CURIE prefix (e.g. 'rdf', 'gs1' followed by a colon followed by a local part. Sites such as https://prefix.cc make it easy to lookup such CURIE prefixes. Linked Data 'documents' declare the CURIE prefixes used within the document and the corresponding URI stems.

The example in Figure 7 also shows that GS1 Digital Link URIs are not only a Web-friendly representation of GS1 identifiers for linking to information and services on the Web.

GS1 Digital Link URIs can also be used in machine-interpretable Linked Data to express facts and relationships about the identified things in a way that can be interpreted by Web search engines and other software. GS1 Digital Link URIs can be used with Linked Data properties defined in the GS1 Web vocabulary, schema.org and other Linked Data vocabularies to express facts about the things they identify.

The W3C RDF standard [RDF] defines a number of useful terms, including:

rdf:Property	Each Linked Data property is a member of the rdf:Property class

rdf:type	Links the Subject to one or more classes to which it belongs
rdf:langString	A literal data type corresponding to a language-tagged string, to support multi-lingual values.

The W3C RDF Schema [RDFS] standard defines some further useful terms, including:

rdfs:Class	Each Linked Data class is a member of the rdfs:Class class
rdfs:domain	Links a property to its domain (the class in which it is defined)
rdfs:range	Links a property to its range (the expected value type)
rdfs:label	Is used to express a label or name for any term (class, property, code list, code list value etc.) within a Linked Data vocabulary or semantic ontology
rdfs:comment	Is used to express a description or definition for any term (class, property, code list, code list value etc.) within a Linked Data vocabulary or semantic ontology
rdfs:subClassOf	Links from a subclass to one of its superclasses
rdfs:subPropertyOf	Links from a subproperty to one of its superproperties

The W3C Web Ontology Language [OWL] standard defines some further useful terms, including:

owl:Class	Each Linked Data class is a member of the owl:Class class
owl:DatatypeProperty	Each Linked Data property that expects to a data value such as a string, language-tagged string, date, date+time, integer. A floating-point value is a member of the owl:DatatypeProperty class
owl:ObjectProperty	Each Linked Data property that expects a complex data object or class or individual Linked Data resource is a member the owl:DatatypeProperty class
owl:Thing	The most general class of things. Most top-level classes within a Linked Data vocabulary are defined to be an rdfs:subclassOf owl:Thing
owl:unionOf	Used to express the class that is the union ('OR') combination of two or more classes
owl:intersectionOf	Used to express the class that is the intersection ('AND') combination of two or more classes

The W3C Simple Knowledge Organization System [SKOS] standard is used for describing concepts and relationships between these concepts. SKOS has the often useful flexibility to describe conceptual relationships (narrower, broader, related etc.) that are looser than formal sub-property or sub-class relationships.

The GS1 Web vocabulary makes use of RDF, RDFS, OWL and SKOS to express its classes, subclasses, properties, code lists and also cross-references and relationships to related terms in other Linked Data vocabularies such as schema.org.

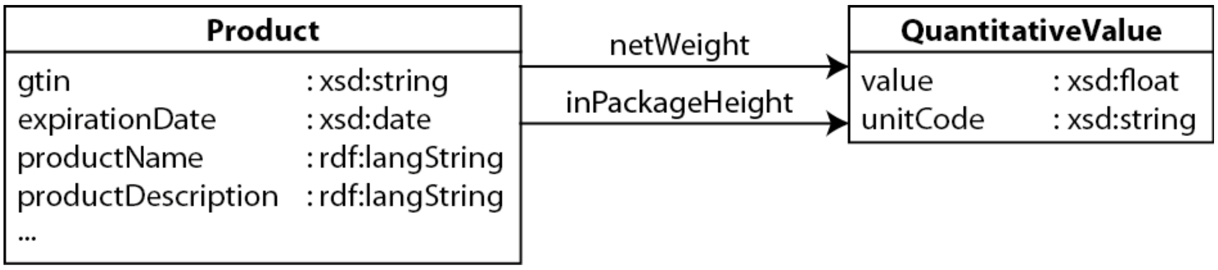
schema.org also defines some useful properties including:

schema:domainIncludes	Links a property to one or more classes in which it is defined
schema:rangeIncludes	Links a property to one or more expected value types

`schema:domainIncludes` and `schema:rangeIncludes` overcome some of the cumbersome use of `owl:unionOf` when using `rdfs:domain` or `rdfs:range` to point to multiple classes that define the property or multiple expected value types; when `rdfs:domain` or `rdfs:range` are used with a list or set of classes, the default interpretation is that the domain or range corresponds to the logical intersection ('AND') of all of these, whereas often the logical union ('OR') is what we more often want to express.

3.1 Using Linked Data to make a UML class diagram machine-interpretable

Taking a very simple UML class diagram such as that shown in Figure 1 (repeated here), it is possible to use such fundamental terms from RDF, RDFS and OWL to convert the UML class diagram into a machine-interpretable Linked Data definition of the same data model.



```
gs1:Product  rdf:type  owl:Class  .
gs1:Product  rdf:type  rdfs:Class  .
gs1:Product  rdfs:subClassOf  owl:Thing  .

gs1:gtin  rdf:type  rdf:Property .
gs1:gtin  rdf:type  owl:DatatypeProperty  .
gs1:gtin  rdfs:domain  gs1:Product  .
gs1:gtin  rdfs:range  xsd:string  .

gs1:expirationDate  rdf:type  rdf:Property .
gs1:expirationDate  rdf:type  owl:DatatypeProperty  .
gs1:expirationDate  rdfs:domain  gs1:Product  .
gs1:expirationDate  rdfs:range  xsd:date  .

gs1:productName  rdf:type  rdf:Property .
gs1:productName  rdf:type  owl:DatatypeProperty  .
gs1:productName  rdfs:domain  gs1:Product  .
gs1:productName  rdfs:range  rdf:langString  .
gs1:productName  rdfs:subPropertyOf  schema:name  .

gs1:productDescription  rdf:type  rdf:Property .
gs1:productDescription  rdf:type  owl:DatatypeProperty .
gs1:productDescription  rdfs:domain  gs1:Product  .
gs1:productDescription  rdfs:range  rdf:langString  .
```

```
gs1:QuantitativeValue  rdf:type  owl:Class  .
gs1:QuantitativeValue  rdf:type  rdfs:Class  .
gs1:QuantitativeValue  rdfs:subClassOf  owl:Thing  .

gs1:netWeight  rdf:type  rdf:Property .
gs1:netWeight  rdf:type  owl:ObjectProperty  .
gs1:netWeight  rdfs:domain  gs1:Product  .
gs1:netWeight  rdfs:range  gs1:QuantitativeValue  .

gs1:inPackageHeight  rdf:type  rdf:Property .
gs1:inPackageHeight  rdf:type  owl:ObjectProperty  .
gs1:inPackageHeight  rdfs:domain  gs1:Product  .
gs1:inPackageHeight  rdfs:range  gs1:QuantitativeValue  .

gs1:value  rdf:type  rdf:Property .
gs1:value  rdf:type  owl:DatatypeProperty  .
gs1:value  rdfs:domain  gs1:QuantitativeValue  .
gs1:value  rdfs:range  xsd:float  .

gs1:unitCode  rdf:type  rdf:Property .
gs1:unitCode  rdf:type  owl:DatatypeProperty  .
gs1:unitCode  rdfs:domain  gs1:QuantitativeValue  .
gs1:unitCode  rdfs:range  xsd:string  .
```

4 Profiles and validation of a data model

There is also a need to be able to validate data, to check that all mandatory data is present and that the data values are expressed using the appropriate data types, e.g. to reject a date value where an integer or floating-point value was expected. Validation schema (often expressed in XSD, JSON Schema or SHACL or Open API interfaces) enable validation to be performed automatically.

It's important to note that different validation schemas exist independently of the data model itself. One can validate a single XML document against multiple XML Schemas which may vary. Therefore it's useful to think of validation as a layer on top of the data model. These layers are called profiles. In the GS1 community, one can imagine a profile for a sector like consumer packaged goods being very different from a profile for pharmaceuticals, and yet they may both selectively use terms from the same data models.

Cardinality constraints indicate the minimum and maximum number of values a property or attribute can express. They are often not intrinsic characteristics of the attribute but depend on the context or profile in which it is used. In contrast, a Linked Data property or attribute can be declared to be a functional property if it logically, semantically can only have one value, such as the date of manufacture of a product for a particular product instance.

XML Schema Definition language (XSD) is concerned with validation of a structured document in which each element must appear within a well-defined sequence and where special provision must be made for user extensions. This is sometimes referred to as a 'closed shape' validation.

JSON Schema and SHACL are concerned with validation of a 'data graph' rather than a rigidly defined document structure of XML. A data graph is typically a tree structure and might visually resemble a 'mind map' in which ovals represent concepts or data classes and the arrows that connect these correspond to defined attributes/properties that express specific relationships.

Compared with XML, it is much easier to add user extensions to a data graph. JSON Schema and SHACL are often configured to do 'open shape' validation in which they simply ignore any user extensions that they were not expecting to validate.

The following table provides a comparison of how some simple validation constraints can be expressed in XSD, JSON Schema and SHACL.

Validation Rule	XSD	JSON Schema	SHACL
Applies to named field	name="*fieldname*" type="*defined_type*"	Fieldname appears within list of "properties"	sh:path fieldname
Mandatory Field	minOccurs="1" (**may be omitted** since default values of minOccurs="1")	Include fieldname within the list of "required" properties	sh:minCount 1 (**must be asserted** since default value of sh:minCount=0)
Note on default values for minOccurs, maxOccurs and sh:minCount, sh:maxCount	*default of XSD minOccurs = 1, maxOccurs = 1)*	*Only properties specified within "required" are considered mandatory*	*default of sh:minCount = 0, default of sh:maxCount = unbounded*
Optional Field	minOccurs="0" (**must be asserted** since default value of minOccurs="1")	Include fieldname within the list of "properties" but omit from list of "required" properties	sh:minCount 0 (**may be omitted** since default value of sh:minCount=0)
Field expects a string	type="xsd:string"	"type":"string"	sh:datatype xsd:string
Field expects a dateTime	type="xsd:dateTime"	"type":"string", "format":"date-time"	sh:datatype xsd:dateTime

Validation Rule	XSD	JSON Schema	SHACL
Field expects an integer	type="xsd:int"	"type":"integer"	sh:datatype xsd:int
Field expects a decimal value	type="xsd:decimal"	"type":"number"	sh:datatype xsd:decimal
Field expects a floating-point value	type="xsd:float"	"type":"number"	sh:datatype xsd:float
Field expects a URI	type="xsd:anyURI" or references an xsd:simpleType with an xsd:restriction base="xsd:anyURI"	"type":"string", "format":"uri"	sh:nodeKind sh:IRI

Field expects a string from a restricted list of enumerated values	references an xsd:simpleType with an xsd:restriction base="xsd:string" containing xsd:enumeration child elements that express the permitted values
	
	e.g.
	<xsd:simpleType name="ActionType">
	    <xsd:restriction base="xsd:string">
	        <xsd:enumeration value="ADD"/>
	        <xsd:enumeration value="OBSERVE"/>
	        <xsd:enumeration value="DELETE"/>
	    </xsd:restriction>
	</xsd:simpleType>
		"type":"string",
		"enum":[…]
		
		e.g.
		"type":"string",
		"enum":[
		"ADD",
		"OBSERVE",
		"DELETE"
		]
			sh:datatype xsd:string
			sh:in ("ADD" "OBSERVE" "DELETE")
			
			e.g.
			epcis:Action_TypeAndFormat owl:sameAs [
			    sh:path epcis:action ;
			    sh:datatype xsd:string ;
			    sh:name "action" ;
			    sh:in epcis:ActionEnum
			].
			
			epcis:ActionEnum owl:sameAs
			("ADD" "OBSERVE" "DELETE").

5 References

CURIE = Compact URI Expressions

- https://www.w3.org/TR/curie/

GS1 Architecture Principles

- Linked from https://www.gs1.org/standards/gs1-architecture

JSON = JavaScript Object Notation

- https://tools.ietf.org/html/rfc8259

JSON-LD = JavaScript Object Notation for Linked Data

- https://www.w3.org/TR/json-ld/

JSON Schema

- https://json-schema.org/

Open API = Open Application Programming Interfaces

- https://www.openapis.org/

OWL = Web Ontology Language

- https://www.w3.org/TR/owl2-overview/
- https://www.w3.org/2002/07/owl#

RDF = Resource Description Framework

- https://www.w3.org/TR/rdf11-concepts/
- https://www.w3.org/TR/rdf11-primer/
- https://www.w3.org/1999/02/22-rdf-syntax-ns#

RDFS = Resource Description Framework Schema

- https://www.w3.org/TR/rdf-schema/

SHACL = Shapes Constraint Language

- https://www.w3.org/TR/shacl/

SKOS = Simple Knowledge Organization System

- https://www.w3.org/TR/skos-primer/
- http://www.w3.org/2004/02/skos/core#

UML = Unified Modelling Language

- https://www.omg.org/spec/UML/

XML = eXtensible Markup Language

- https://www.w3.org/TR/REC-xml/

XSD = XML Scheme Definition language

- https://www.w3.org/TR/xmlschema-1/
- https://www.w3.org/TR/xmlschema-2/

Contributors to this Technical Bulletin

Name	Organisation
Dr Mark Harrison	Consultant to GS1 Global Office
Phil Archer	Director, Web Solutions, GS1 Global Office